UNIVERSIDAD CARLOS III DE MADRID

DESARROLLO DE UNA HERRAMIENTA GRÁFICA PARA GESTIÓN DE PROYECTOS

ESCUELA POLITÉCNICA SUPERIOR
INGENIERÍA TÉCNICA DE TELECOMUNICACIONES: ESPECIALIDAD TELEMÁTICA
PROYECTO FIN DE CARRERA

Autor: Fernando Flores Redondo
Tutor: Francisco Valera Pintor
Fecha: 9 de Septiembre 2013

Índice

Índice de figuras

AGRADECIMIENTOS

A Fernando y Yolanda, Yolanda y Fernando,
tanto monta monta tanto;
A ti, que llegaste amarilla
y te convertiste en mi mejor pesadilla;
A você, menina linda,
obrigado por estar aqui ainda.
Casas pequeñas, antes y ahora.
Muchas natillas; y La Concha de la Lora.

1 INTRODUCCIÓN

1.1 Descripción del trabajo realizado

La presente memoria describe en profundidad el trabajo realizado en cada una de las fases (definición, análisis, implementación) así como los distintos procedimientos y herramientas utilizados en el devenir del proyecto "DESARROLLO DE UNA HERRAMIENTA GRÁFICA PARA GESTIÓN DE PROYECTOS".

Centrándonos en la naturaleza del proyecto, una primera descripción a alto nivel nos permite hacer hincapié en el hecho de que, a partir de una necesidad de negocio específica descrita y publicada por el Departamento de Ingeniería Telemática de la Universidad Carlos III de Madrid a través del Profesor Francisco Valera, se propone de forma formal la realización de este Proyecto, cuya implementación satisfará dicha necesidad de negocio y cuyo resultado final, incluyendo documentación y presentación ante un Tribunal de Evaluación, será presentado como Proyecto Final de Carrera para el alumno que suscribe estas líneas.

1.2 Objetivos

- Proporcionar una versión totalmente funcional de la herramienta, sin funcionalidades experimentales, para lo cual se hace imprescindible su validación con un proyecto real.
- El resultado final no será un prototipo, sino una versión comercial para ser utilizada en proyectos en los que la Universidad Carlos III participe junto con otras Universidades europeas.
- Permitir que la herramienta permita futuras mejoras y/o ampliaciones en sus funcionalidades sin afectar al desarrollo existente.
- Finalizar el desarrollo antes de la fecha límite especificada (Octubre de 2013).

1.3 Motivación

El objetivo es el desarrollo de una aplicación colaborativa, con acceso vía web, que facilite la gestión automática de generación de informes de actividad en proyectos. La herramienta estará especialmente orientada a la gestión de proyectos financiados por la Comisión Europea.

Estos proyectos tienen una serie de características que definen en gran medida cómo la herramienta ha de ser implementada, a saber:

- Los proyectos se dividen en paquetes de trabajos (Workpackages, WPs), cada uno de los cuales puede constar a su vez de una o más tareas.
- Cada proyecto tiene un coordinador.
- Cada Workpackage tiene un líder.
- Cada Workpackage tiene asignados uno o más partners, que trabajan en él.

- Se definen en cada proyecto distintas fechas para la realización de informes de trabajo; cada partner ha de realizar un informe en cada una de esas fechas par cada uno de los WPs en los que participa.
- Cada informe de trabajo ha de contener información sobre el trabajo realizado y los resultados obtenidos en el período de tiempo al que se refiere el informe, el esfuerzo de cada partner, y los gastos en los que ha incurrido.

Esta herramienta no es un gestor de proyecto al uso (cuyo exponente más conocido y utilizado puede ser el programa Microsoft Project), ya que no está orientada a ser usada única y exclusivamente por el gestor del proyecto (Project Manager); la diferencia principal radica en su anteriormente mencionada naturaleza colaborativa, que permite que todos los miembros del proyecto accedan a ella e informen del estado (y resultado) de su trabajo, quedando almacenada dicha información de forma centralizada y visible para aquellos participantes que cuenten con permisos para ello.

Existen diversos productos que podrían ser utilizados para este fin; la herramienta actualmente en el mercado que mejor se adecúa a las necesidades sería el módulo Reporter, del paquete Eurestools, que cubre las necesidades definidas pero cuyo coste de licencia (1.800€ por año para un proyecto de 10 partners, más un 4% adicional por cada partner extra) suponen un desembolso importante, de modo que nuestra herramienta, de uso gratuito, supondrá un ahorro considerable.

En la actualidad, el departamento de Telemática de la Universidad Carlos III está coordinando uno de estos proyectos de investigación europeos (*Trilogy 2*). Se está haciendo uso de una herramienta colaborativa (del tipo “wiki) para realizar estas tareas de gestión y seguimiento de proyectos; no existen en tal opción ningún tipo de procesos automáticos de generación de informes, notificaciones o seguimiento, lo que hace de ella una alternativa poco óptima y configurable.

1.4 Estructura de la memoria

Este documento está estructurado en cuatro partes claramente diferenciadas:

- Gestión del proyecto (Capítulo 2): Descripción y resultado del trabajo y herramientas de análisis relativos a los procedimientos seguidos durante la ejecución del proyecto.
- Análisis y diseño (Capítulos 3 y 4): Definición detallada de requisitos funcionales.
- Implementación y validación (Capítulos 5 y 6): Descripción del trabajo realizado a nivel técnico.
- Cierre (Capítulo 7): Lecciones aprendidas durante el desarrollo del proyecto, próximos pasos y conclusiones.

1.5 Glosario de términos

A pesar de que todos estos términos son explicados a los largo del documento, vale la pena agruparlos en este apartado para que su comprensión no requiere de una búsqueda exhaustiva en el propio documento.

1.5.1 Términos relativos a proyectos e informes

- WP: Del inglés Workpackage, paquete de trabajo. Estas siglas son utilizadas a lo largo de todo el documento, incluyendo su variante en plural, WPs.
- Partner: Miembro del proyecto sin responsabilidades de administración. Este término es utilizado en este documento, así como su variante plural, partners.
- Informe: Resultados del trabajo de todos partners envueltos en un determinado proyecto. Se divide en sub informes.
- Sub informe: Parte de un informe en la que un determinado partner detalla el trabajo realizado para un determinado WP en un período determinado.

1.5.2 Siglas y abreviaturas técnicas

- XML: siglas en inglés de eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas utilizado para almacenar datos en forma legible.
- Javascript: JavaScript es un lenguaje de programación interpretado, orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico que se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web.
- AJAX (Asynchronous JavaScript And XML): Técnica de desarrollo web para crear aplicaciones interactivas, que se ejecutan en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano.
- jQuery: Librería de funciones JavaScript que facilita, entre otros, el acceso a los elementos del DOM, la interacción con documentos HTML y XML y lainteracción con la tecnología AJAX.
- HTTP: Http son las siglas de HyperText Transfer Protocol, el método utilizado para transferir ficheros hipertexto por Internet.
- URL (sigla en inglés de Uniform Resource Locator), es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación.
- REST: La Transferencia de Estado Representacional (Representational State Transfer) define un conjunto de principios arquitectónicos por los cuales se diseñan servicios web haciendo hincapié en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes.
- API: (Application Program Interface). Conjunto de convenciones que definen cómo debe invocarse una determinada función/recurso de un programa desde una aplicación.

2 GESTIÓN DEL PROYECTO: PLAN DE EJECUCIÓN

2.1 Introducción: Técnicas de Gestión de Proyectos

Dada la complejidad del proyecto que nos atañe, se ha optado por recurrir a las técnicas de gestión de proyectos actualmente más aceptadas por la industria del desarrollo de software.

Desde las más tempranas fases de análisis hasta los pasos finales, este proyecto ha utilizado como referencia la [21] *Guía de los fundamentos de la dirección de proyectos* (más conocida como PMBOK por sus siglas en inglés), el estándar más ampliamente reconocido para manejar y administrar proyectos.

Desarrollada por el Project Management Institute (PMI), esta guía es el conjunto de conocimientos en Dirección/Gestión/Administración de Proyectos generalmente reconocidos como “buenas prácticas”, y que se constituye como estándar de facto en la Administración/Gestión de proyectos.

Antes de aplicar esta serie de técnicas, se hace imprescindible comprobar que nuestro trabajo se ajusta a la definición de Proyecto aportada por la Guía PMBOK:

"*Un proyecto es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único en un tiempo determinado."*

Analicemos los dos conceptos clave de esta definición:

- Temporal: El proyecto tiene un comienzo y un final definido.
- Productos, servicios o resultados únicos: Un proyecto crea productos entregables únicos. La singularidad es una característica importante de los productos entregables de un proyecto.

Ambas condiciones se cumplen en el caso de este Proyecto, por lo que consideramos que la aplicación de las enseñanzas de la Guía PMBOK pueden ser perfectamente aplicadas a nuestro trabajo en el mismo.

La gestión de un proyecto incluye:

- Identificación de requisitos
- Establecimiento de objetivos claros y realizables
- Equilibrio de las demandas concurrentes de calidad, alcance, tiempo y costes
- Adaptación a las especificaciones, los planes y el enfoque a las diversas necesidades y expectativas de todas las partes involucradas.

Generalmente se habla de una “triple restricción” —alcance (scope), tiempos (time) y costes (cost) del proyecto (ver gráfica siguiente) — a la hora de gestionar los

requisitos de un proyecto, siendo la calidad (quality) la “cuarta dimensión” en este concepto.

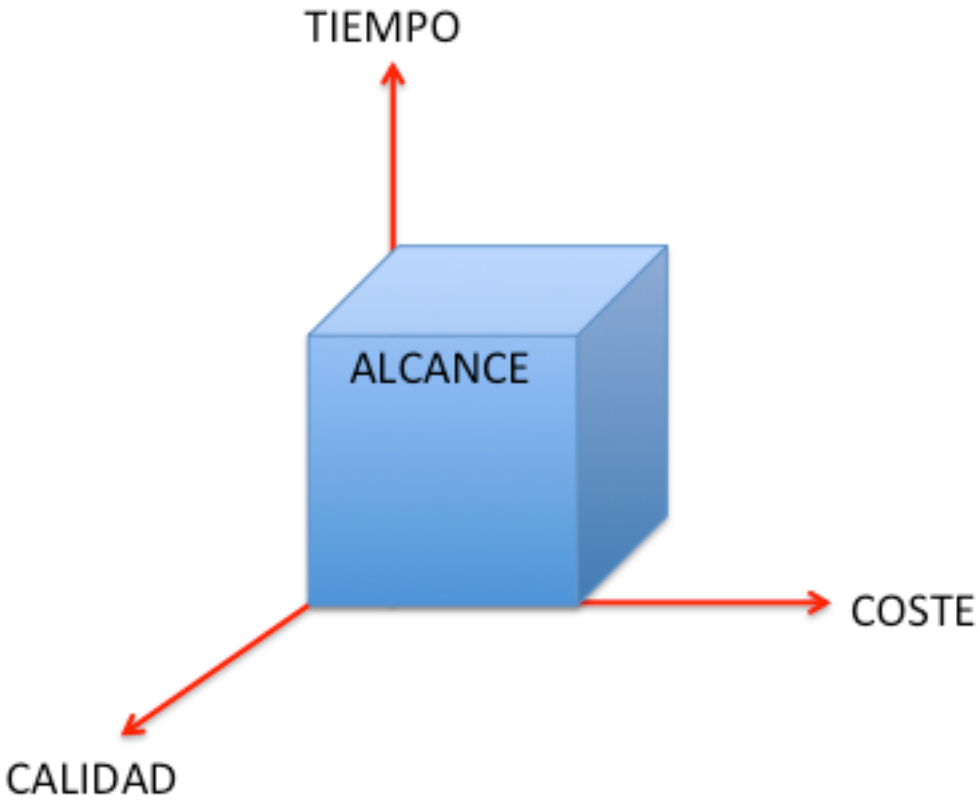


Figura 1 – Triple restricción en gestión de proyectos

Es evidente que esta triple restricción obliga a priorizar los tres factores envueltos en la fórmula, y que, de forma conjunta definen el alcance del proyecto.

En nuestro caso se han priorizado la calidad, entendida como la satisfacción de los requerimientos del sponsor, y el tiempo, dado que se definió una fecha tope e inamovible para la entrega del proyecto. Dadas las especiales características del proyecto (proyecto académico con un sólo desarrollador), el coste no tiene el peso tan importante como en aquellos proyectos comerciales al uso, en los que concurren los intereses de más partes.

Teniendo todos estos aspectos iniciales en consideración, se procede a detallar todos y cada uno de los puntos que definen cómo se ha gestionado este proyecto, no sin antes mencionar que los apartados 2.3 y 2.5 se corresponden respectivamente con los apartados 1.3 y 1.2 del capítulo de Introducción; se repiten de forma casi literal aquí para, junto con el resto de apartados, tener una descripción completa del proyecto.

2.2 Roles en el proyecto

Como se ha mencionado en el apartado anterior, este proyecto tiene unas características especiales que hacen que el personal involucrado se reduzca al mínimo:

- Patrocinador y Tutor: Francisco Valera, Profesor de la Universidad Carlos III
- Desarrollador: Fernando Flores, alumno de la Universidad Carlos III

2.3 Necesidad de negocio

El objetivo es el desarrollo de una aplicación que facilite la gestión automática de generación de informes de actividad en proyectos. La herramienta estará especialmente orientada a la gestión de proyectos financiados por la Comisión Europea.

Existen diversos productos que podrían ser utilizados para este fin, pero ninguno de ellos cumple las expectativas específicas que se buscan en este proyecto, con el añadido del coste por mantenimiento en el que se incurriría en caso de optar por alguna de ellas.

En la actualidad, se está haciendo uso de una herramienta colaborativa (del tipo "wiki) para realizar estas tareas de gestión y seguimiento de proyectos, siendo ésta una alternativa poco óptima y poco configurable.

2.4 Descripción del producto

Para satisfacer la necesidad de negocio anteriormente descrita, se decide desarrollar una aplicación web que dé cabida a todos los requerimientos inherentes a la definición del proyecto. Véase:

- Creación de proyectos (compuestos de Paquetes de Trabajo y Tareas)
- Creación de partners
- Asignación de tareas y paquetes de trabajo a los distintos miembros del proyecto
- Generación de informes de trabajo
- Posibilidad de aprobar o denegar dichos informes y aportar comentarios que justifiquen dichas acciones
- Envío dichos reportes por correo electrónico
- Automatización del proceso de notificaciones relativo a fechas de entrega de proyectos

2.5 Objetivos

- Proporcionar una versión totalmente funcional de la herramienta, sin funcionalidades experimentales, para lo cual se hace imprescindible su validación con un proyecto real.
- El resultado final no será un prototipo, sino una versión comercial para ser utilizada en proyectos en los que la Universidad Carlos III participe junto con otras Universidades europeas.
- Permitir que la herramienta permita futuras mejoras y/o ampliaciones en sus funcionalidades sin afectar al desarrollo existente.
- Finalizar el desarrollo antes de la fecha límite especificada (Octubre de 2013).

2.6 Alcance

En este proyecto se encarga tanto del desarrollo de la aplicación web como de la lógica de negocio que le da soporte, así como de la documentación técnica y memoria; no se incluyen en él otro tipo de interfaz (como pudiera ser una específica para dispositivos móviles), así como ninguna guía de desarrollo para las tecnologías analizadas.

2.7 Entregables

Definición de entidades
Definición de roles
Modelo de datos
Esquema de relación datos-entidades
Implementación del modelo de datos (esquema)
Gestión de entidades
Gestión de roles
Módulo de notificación
Módulo de envío de informes
Módulo de administración de usuarios
Módulo de administración de proyectos
Módulo de administración de informes
Guía de usuario
Documentación técnica
Memoria general del proyecto

Tabla 1: Entregables del proyecto

2.8 Restricciones

El desarrollo de este proyecto cuenta con varias circunstancias que en cierto modo pudieran llegar a afectar el normal devenir del mismo. Se han identificado como claves las siguientes:

- Localización geográfica: Desarrollador y patrocinador se encuentran en continentes distintos, con una diferencia horaria de 6 horas, lo que imposibilita el contacto físico directo.
- Imposibilidad de dedicación a tiempo completo: Dadas las circunstancias personales/profesionales, el desarrollador no puede dedicarse al proyecto con dedicación exclusiva, lo que puede redundar en retrasos en los plazo establecidos.

2.9 Suposiciones

Se han dado por hecho los siguientes supuestos:

- El alumno/desarrollador conoce los lenguajes y herramientas necesarias para realizar este proyecto
- El profesor/sponsor acepta las restricciones mencionadas en el punto anterior
- El entorno de ejecución final, proporcionado por la Universidad, satisfará las necesidades técnicas de la implementación final del proyecto.

2.10 Matriz de Trazabilidad de Requerimientos

Es esta una herramienta en la que se vinculan los requerimientos del proyecto con sus entregables correspondientes, asegurando que los primeros son cubiertos y especificando qué entregable es el que cubre cada uno de ellos. Además, se especifica la fase del proyecto (en base a la Estructura de Descomposición del Trabajo, ver siguiente apartado) en la que se entrega cada parte.

	Requerimiento	Entregable	WBS#
A	Definir un proceso para gestionar el envío de reportes	Identificación de entidades y roles	2.1
		Casos de uso	2.2
B	Crear una estrucutra de datos que respalde dicho proceso	Definición de arquitectura	2.3
		Modelo de datos	2.4
		Implementación del modelo de datos (esquema)	2.4
C	Desarrollar una lógica de negocio para la gestión de los datos	Adaptador de recursos	3.1
		Gestión de entidades y roles	3.2,3.3
D	Desarrollar herramientas para comunicación de la información	Módulo de notificación	3.4
		Gestor de envío de informes	3.4
E	Implementar una interfaz Web	Módulo de administración de usuarios	2.3, 2.4, 4.1
		Módulo de administración de proyectos	2.3, 2.4, 4.2
		Módulo de gestión de informes	2.3, 2.4, 4.3
F	Documentar el proceso	Guía de usuario	5.1
		Documentación del API	5.2
		Memoria general del proyecto	5.3

Tabla 2: Matriz de trazabilidad de requerimientos

2.11 Estructura de Descomposición del Trabajo

La Estructura de Descomposición o Desglose del Trabajo (EDT, o WBS en inglés, siglas de Work Breakdown Structure) es una descomposición jerárquica, orientada al producto final entregable del trabajo que será ejecutado por el equipo del proyecto, para lograr los objetivos del proyecto y crear los productos entregables requeridos. Organiza y define el alcance total al subdividir el trabajo en porciones de trabajo más pequeñas y fáciles de manejar, llamados paquetes de trabajo, que pueden programarse, costearse, supervisarse y controlarse.

Se presenta aquí la descomposición del trabajo para este proyecto, con una posterior reseña breve de cada parte.

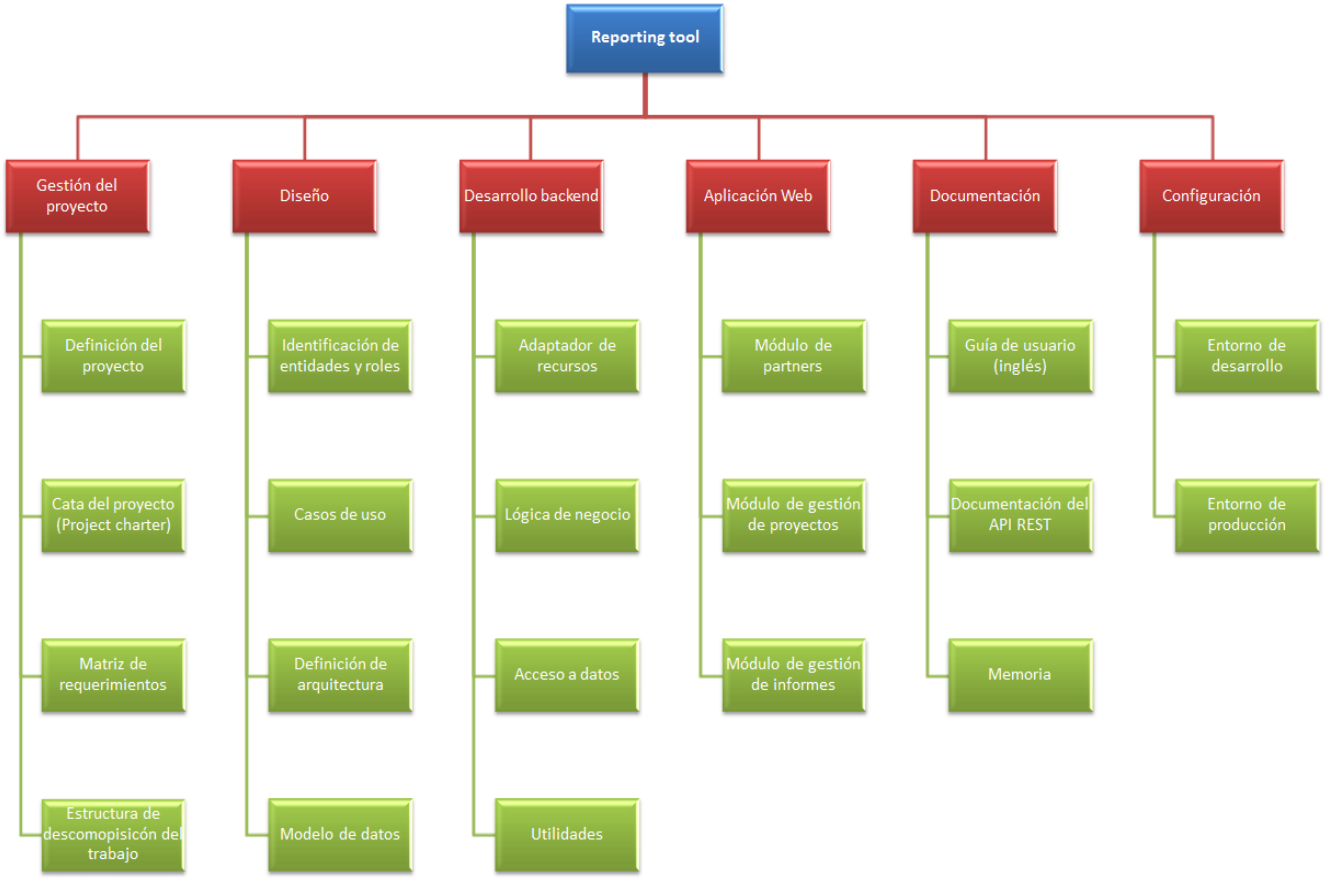


Figura 2: Estructura de Descomposición del trabajo

Representación numérica:

0. Herramienta de gestión de informes
1. Gestión del proyecto
   1.1. Definición del proyecto
   1.2. Cata del proyecto (Project charter)
   1.3. Matriz de requerimientos
   1.4. Estructura de descomopisicón del trabajo

2. Diseño
    2.1. Identificación de entidades y roles
    2.2. Casos de uso
    2.3. Definición de arquitectura
    2.4. Diseño de interfaz web
3. Desarrollo backend
    3.1. Adaptador de recursos
    3.2. Lógica de negocio
    3.3. Acceso a datos
    3.4. Utilidades
4. Aplicación Web
    4.1. Módulo de partners
    4.2. Módulo de gestión de proyectos
    4.3. Módulo de gestión de informes
5. Documentación
    5.1. Guía de usuario (inglés)
    5.2. Documentación del API REST
    5.3. Memoria
6. Configuración
    6.1. Entorno de desarrollo
    6.2. Entorno de producción

3 ANÁLISIS DE REQUISITOS FUNCIONALES

3.1 Definición de entidades

3.1.1 Proyecto

- Conjunto de actividades a realizar. Engloba al resto de entidades. Es creado por el Administrador.
- Su estado cambia a través del tiempo dependiendo de las interacciones con los distintos actores que forman parte de la herramienta:
  - PENDING: Cuando el proyecto es creado, se le asigna este estado por defecto. Lo mantendrá hasta que todos sus elementos (WPs, Tareas, Fechas de entrega de informes) sean definidos.
  - ACTIVE: Una vez el proyecto está totalmente definido, pasará al estado activo, de modo que la herramienta lo considerará a la hora de mostrarlo disponible para la edición de informes.
  - CLOSED: Cuando el proyecto finaliza, pasa a este estado, y la herramienta no permitirá editar su información llegados a este punto.

3.1.2 Tarea

- Trabajo que ha de realizarse en un periodo limitado de tiempo.

3.1.3 WP

- Contiene una o varias tareas, y un líder.

3.1.4 Partner

- Persona/grupo que forma parte del proyecto. Puede tener distintos roles. Está asignado a una o más tareas.

3.1.5 Informe

- Informe de trabajo para un determinado partner en un determinado periodo de tiempo. Puede incluir uno o más sub Informes.

3.1.6 Sub Informe

- Informe de trabajo para un determinado partner en una determinada tarea.

3.2 Definición de roles y sus requisitos funcionales

3.2.1 Administrador

- Acceder a todas las funcionalidades de la herramienta
- Crear nuevos proyectos y editar los existentes
- Asignar un coordinador a cada proyecto

3.2.2 Coordinador

- Tener acceso a todos los subinformes del proyecto que coordina

- Tener visión directa de los estados críticos del proyecto (banderas rojas)
- Tener visión directa del esfuerzo por WP
- Poder enviar subinformes por correo electrónico
- Poder editar cualquier informe del proyecto que coordina

3.2.3 Líder de WP

- Tener acceso a todas las tareas del WP
- Tener acceso a los subinformes relativos a las tareas incluidas en el WP
- Tener visión directa del estado y los detalles de estos subinformes
- Validar los subinformes de los partners
- Enviar subinformes por correo electrónico

3.2.4 Partner

- Tener visión directa de los informes en los que está involucrado
- Tener visión directa de los WPs en los que está involucrado
- Tener visión directa del esfuerzo realizado hasta ahora
- Tener visión directa del esfuerzo restante
- Tener acceso a la edición del status de cada uno de sus informes
- Tener acceso a la edición detallada de cada uno de los informes
- Poder enviar informes por correo electrónico

3.3 Flujo de trabajo

Descripción gráfica a alto nivel del flujo de trabajo desde que un proyecto es creado hasta que los informes son generados.

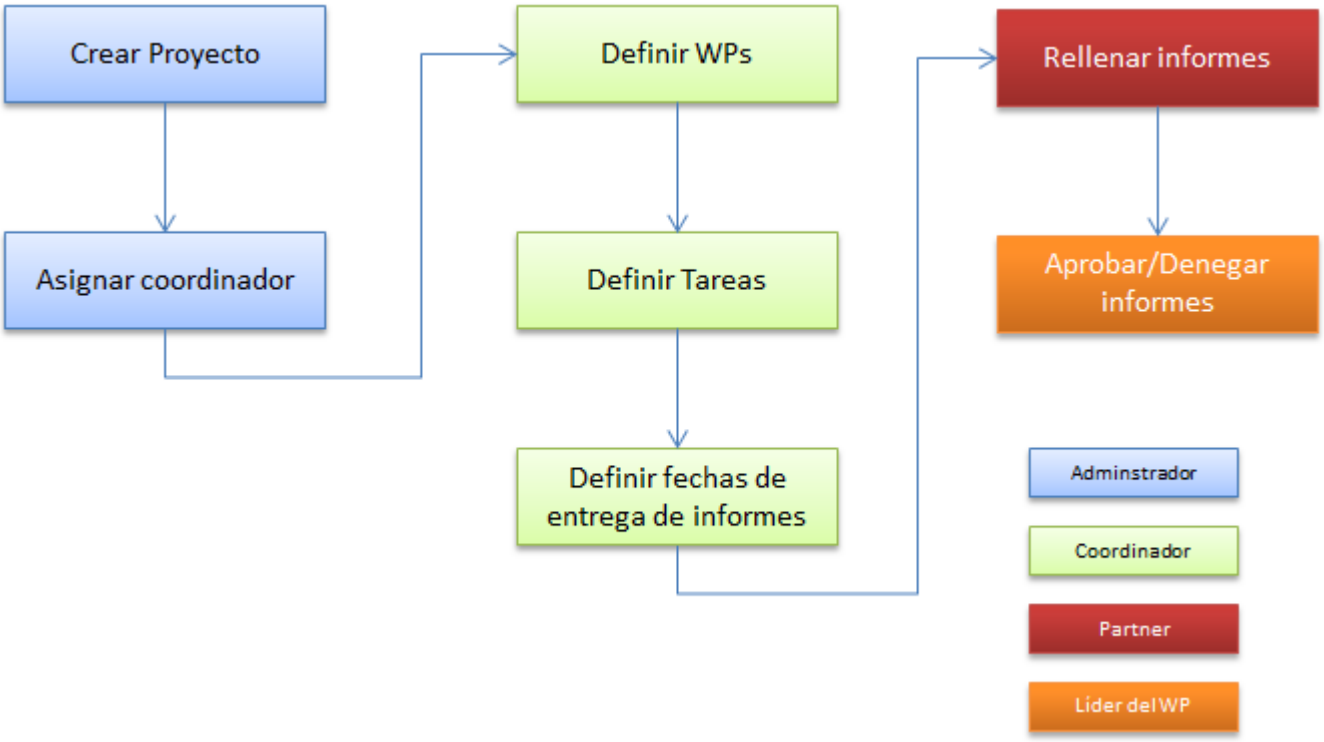


Figura 3: Flujo de trabajo en la herramienta

3.4 Casos de uso

A continuación se detallan los distintos casos de uso de la herramienta, incluyendo, cuando proceda, una captura de pantalla a modo de ilustración del caso.

3.4.1 Login

Identificador	1
Caso de uso	Login
Descripción	Acceder a la herramienta a partir de un formulario de identificación en el que cada usuario ha de usar el nombre de usuario y contraseña que le han sido asignados.
Actores (roles)	Todos los usuarios de la herramienta.
Resultado	El usuario accede a la herramienta.
Dependencias	

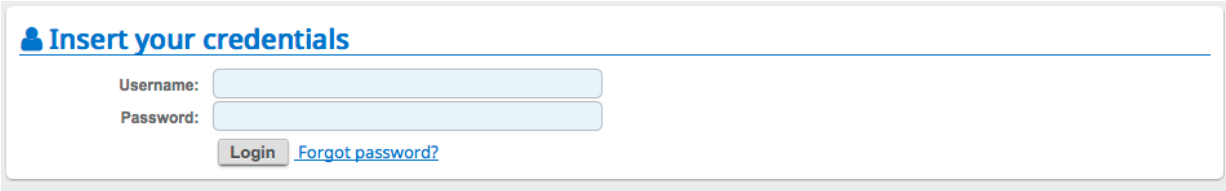


Figura 4: Menú de login

3.4.2 Crear Partner

Identificador	2
Caso de uso	Añadir partner
Descripción	Crear un nuevo partner y, por consiguiente, un nuevo usuario de la herramienta, al que se le asigna un nombre de usuario, una contraseña, una dirección de email y el listado de miembros que se desean incluir. Una vez creado el usuario, la herramienta envía esta información por correo electrónico de forma automática a la dirección especificada.
Actores (roles)	Administrador
Resultado	El partner es dado de alta en la herramienta y por tanto puede acceder a ella.
Dependencias	1

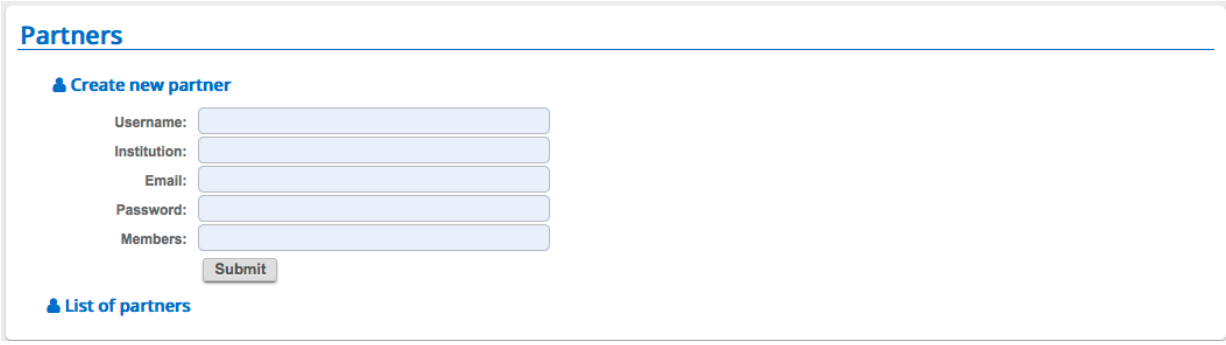


Figura 5: Menú de creación de partner

3.4.3 Recordar contraseña

Identificador	3
Caso de uso	Recordar contraseña
Descripción	Cuando un usuario accede al formulario de login y no recuerda su contraseña, tiene la posibilidad de recuperarla (le será enviada por correo electrónico); para ello debe proporcionar el nombre de usuario y dirección de email definidas en su cuenta.
Actores (roles)	Todos los usuarios
Resultado	Las credenciales de acceso son enviadas por correo electrónico.
Dependencias	

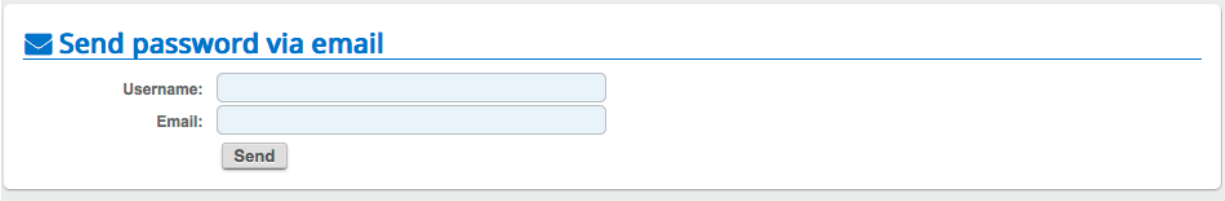


Figura 6: Formulario de reenvío de contraseña

3.4.4 Crear proyecto

Identificador	4
Caso de uso	Creación de un nuevo proyecto
Descripción	El administrador crea un nuevo proyecto con la información básica: Título, descripción, fechas de inicio y fin, y coordinador.
Actores (roles)	Administrador
Resultado	El proyecto queda almacenado en el fichero contenedor de proyectos.
Dependencias	1

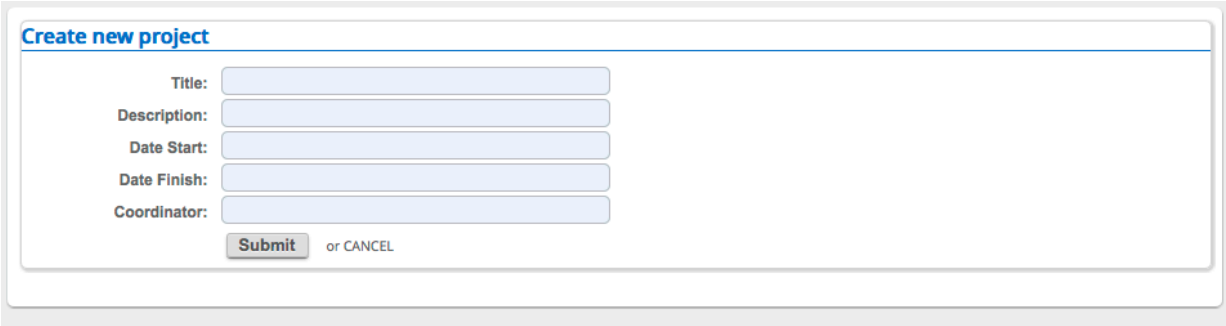


Figura 7: Formulario de creación de proyecto

3.4.5 Definir elementos del proyecto

Identificador	5
Caso de uso	Completar toda la información del proyecto
Descripción	Una vez que el proyecto ha sido creado, se han de definir en él todos los elementos que lo componen: WPs, tareas, fechas de entrega de informes, y partners asociados a los WPs y tareas.
Actores (roles)	Coordinador del proyecto
Resultado	El contenido del proyecto es actualizado en el fichero contenedor de Proyectos.
Sub caso(s)	Este caso de uso comprende varias acciones que se pueden realizar, dependiendo en el elemento a modificar: • Añadir fecha de entrega de informe • Añadir WPs y tareas • Asignar partner(s) a WP y tareas • Añadir esfuerzo para un partner en un WP
Dependencias	1,4

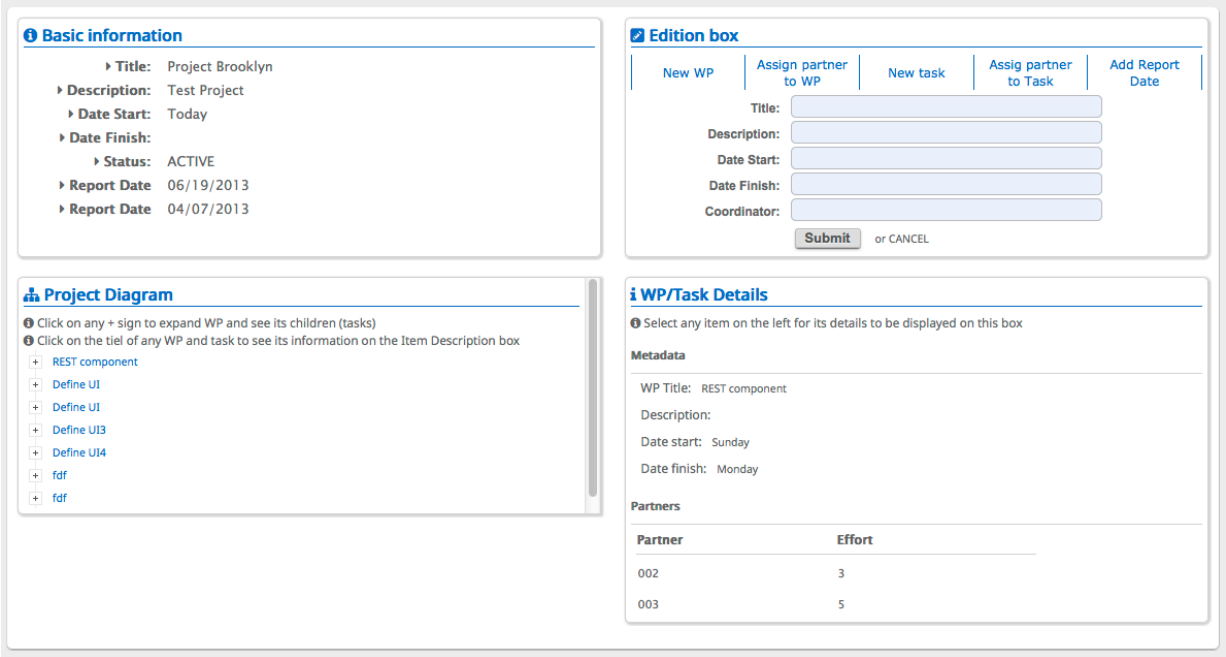


Figura 8: Detalles de un proyecto

3.4.6 Seleccionar proyecto

Identificador	6
Caso de uso	Seleccionar proyecto a editar
Descripción	Se ha de proporcionar una interfaz que posibilite seleccionar qué proyecto de entre los ya creados se quiere editar.
Actores (roles)	Administrador, Coordinador
Resultado	El proyecto, con todos los detalles definidos hasta el momento, le es mostrado al usuario
Dependencias	1,4

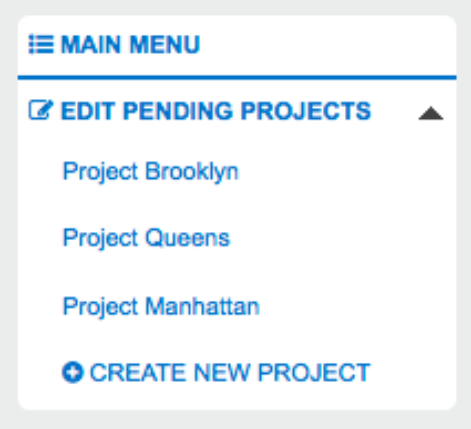


Figura 9: Menú de selección de proyecto

3.4.7 Activar proyecto

Identificador	7
Caso de uso	Cambiar el estado del proyecto a “Activo”
Descripción	Cuando el Administrador crea un proyecto, éste es asignado por defecto con el estado “Pending” (pendiente). Una vez que todos sus elementos están definidos, se debe posibilitar el cambio de ese status a “Active”.
Actores (roles)	Coordinador
Resultado	• El estado del proyecto es actualizado en el fichero contenedor de Proyectos. • Se genera el fichero de que contendrá todos los subinformes para este proyecto
Dependencias	1,4,5

3.4.8 Mostrar proyectos activos

Identificador	8
Caso de uso	Mostrar proyectos activos
Descripción	En la vista de Informes, se mostrarán todos los proyectos activos, que deberán poder ser seleccionados por el usuario de la herramienta
Actores (roles)	Todos los usuarios
Resultado	Todos los proyectos activos son listados
Dependencias	1,7

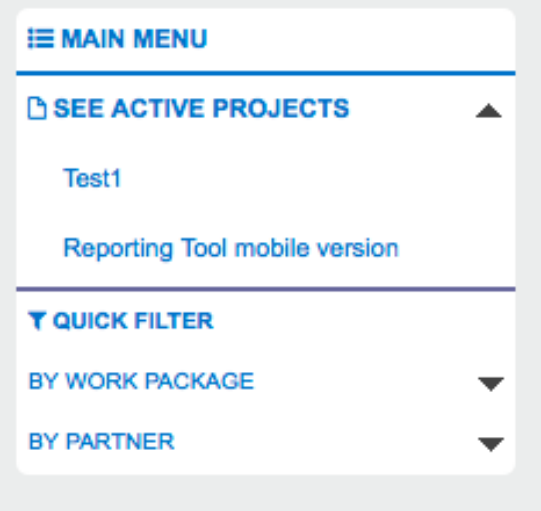


Figura 10: Menú de selección de proyecto activo

3.4.9 Listar subinformes para un proyecto

Identificador	9
Caso de uso	Mostrar todos los subinformes de un determinado proyecto
Descripción	Una vez mostrados todos los proyectos activos, seleccionando uno de ellos se muestran todos los subinformes asociados a él, con un resumen de su información más relevante.
Actores (roles)	Administrador, Coordinador del proyecto, Partner que forme parte del proyecto.
Resultado	• La lista de subinformes es mostrada por pantalla • En caso de que el usuario no forme parte del proyecto en cuestión, un mensaje de error es mostrado y los subinformes no se muestran
Dependencias	1,8

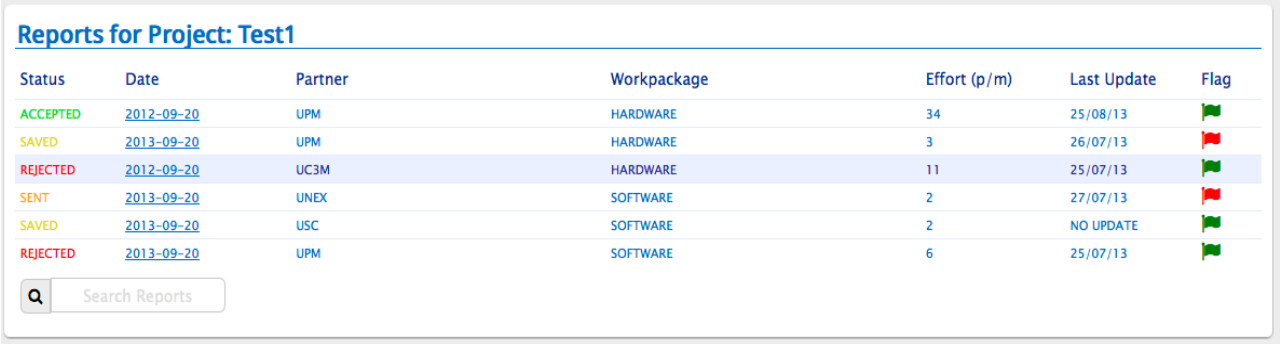


Figura 11: Listado de subinformes de un proyecto

3.4.10 Ver subinforme

Identificador	10
Caso de uso	Visión detallada de un determinado subinforme
Descripción	De la lista de subinformes para un proyecto, el usuario selecciona uno de ellos.
Actores (roles)	Administrador, Coordinador del proyecto, Partner que forme parte del proyecto.
Resultado	Los detalles del informe son mostrados por pantalla.
Dependencias	1,9

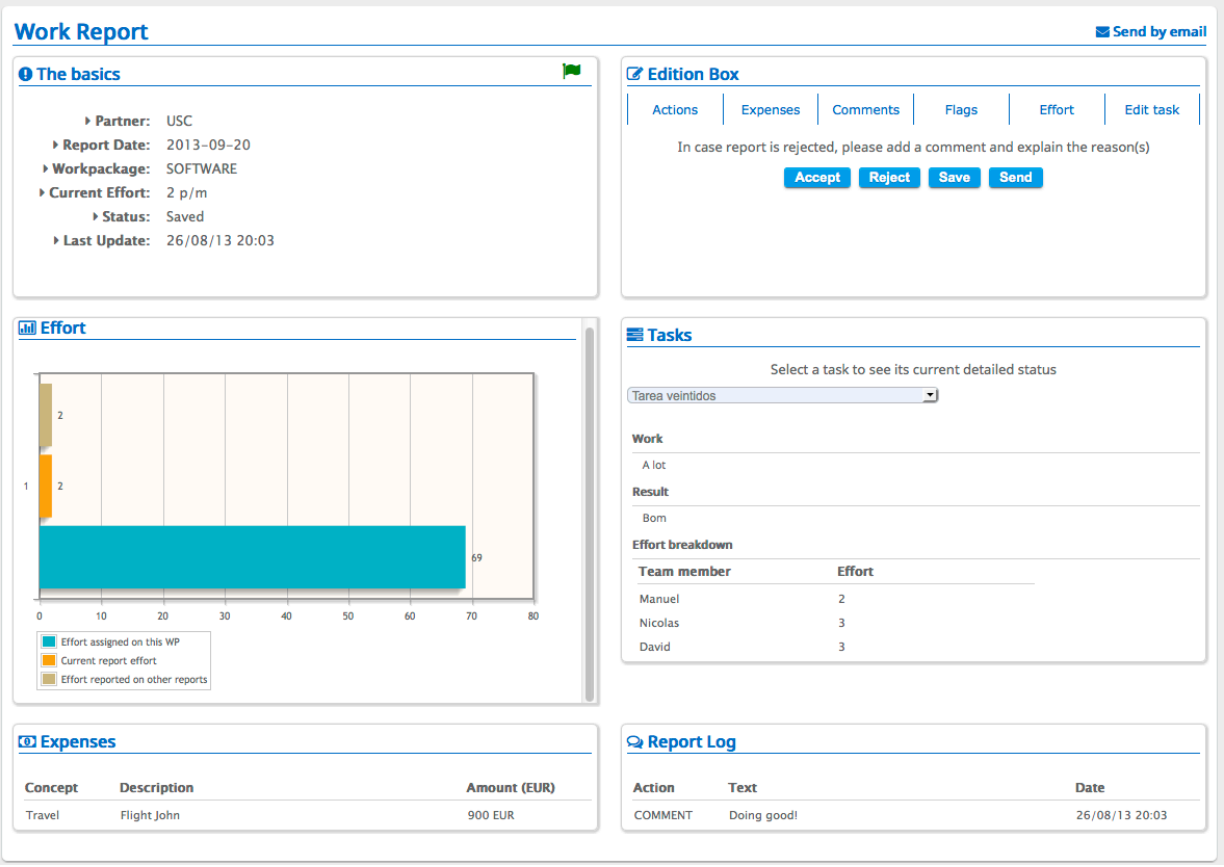


Figura 12: Detalles de un subinforme

3.4.11 Editar subinforme

Identificador	11
Caso de uso	Edición de un determinado subinforme
Descripción	De la lista de subinformes para un proyecto, el usuario selecciona uno de ellos para ser editado.
Actores (roles)	Administrador, Coordinador del proyecto, Partner asignado a ese subinforme.
Resultado	El subinforme es actualizado en el fichero de informe relacionado al proyecto en cuestión. El usuario puede editar las siguientes partes: • Flags • Añadir comentario • Añadir partida de gastos • Justificar esfuerzo • Describir trabajo y resultados para una determinada tarea
Sub caso(s)	Cambiar estado de un subinforme: • El partner asignado puede salvar o enviar el subinforme • El coordinador del proyecto puede aprobar/denegar el subinforme
Dependencias	1,9

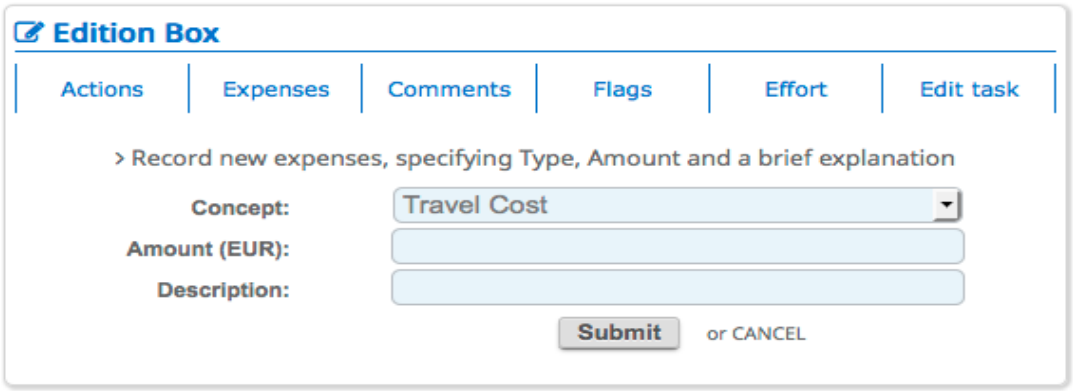


Figura 13: Menús de edición de un subinforme

3.4.12 Enviar subinforme por correo electrónico

Identificador	12
Caso de uso	Enviar subinforme por correo electrónico
Descripción	Un usuario de la herramienta precisa de enviar por correo electrónico el informe que está visualizando a una dirección de correo de su elección.
Actores (roles)	Administrador, Coordinador del proyecto, Partner que forme parte del proyecto.
Resultado	El subinfome es enviado por correo electrónico a la dirección

	especificada por el usuario.
Dependencias	1,9

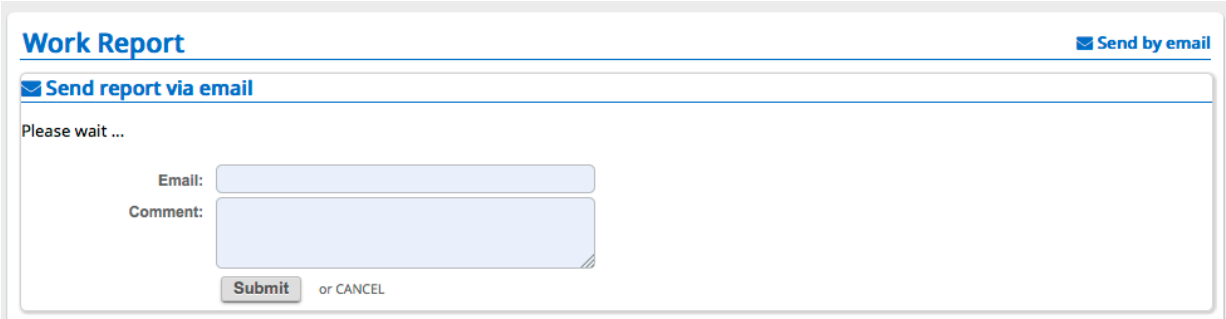


Figura 14: Formulario de envío de subinforme por email

3.4.13 Cerrar proyecto

Identificador	13
Caso de uso	Cerrar proyecto
Descripción	Una vez el proyecto ha finalizado, el estado del proyecto pasa a ser “CLOSED”
Actores (roles)	Administrador, Coordinador del proyecto.
Resultado	El estado del proyecto es actualizado en el fichero contenedor de ficheros.
Dependencias	1

4 DISEÑO DE LA APLICACIÓN

4.1 Objetivo

Se pretende crear una herramienta web que se valga de tecnologías modernas tanto en la interfaz como en el resto de capas de la arquitectura definida.

La última ola de desarrollo de aplicaciones web, que comenzó en la segunda mitad de la pasada década, presenta como estándar de facto la necesidad de que en ellas se proporcionen interfaces de navegador de calidad, tales como aquellos que usan Asynchronous JavaScript and XML (Ajax) o el kit de herramientas Web de Google (GWT); del mismo modo, deben valerse, en caso de necesidad de interacción cliente-servidor, de servicios web RESTful para las aplicaciones cliente externas. No este un requerimiento obligatorio para aplicaciones web de todo tipo, pero en nuestro caso se hace poco menos que imprescindible, dada la naturaleza de la aplicación.

En este proyecto se usa un Resource Request Handler (RRH) (controlador de solicitudes de recursos) para Ajax y para las llamadas de aplicaciones clientes externas. Dicho controlador está ligado a una capa de lógica de negocios, que, a su vez, interactúa con la capa de acceso a datos.

Pasamos a detallar cada parte.

4.2 Arquitectura: Estado del arte y aplicación a este proyecto

4.2.1 Capa de aplicación

4.2.1.1 Introducción: REST

La Transferencia de Estado Representacional (REpresentation State Transfer - REST) describe un estilo arquitectónico de sistemas en red como, por ejemplo, aplicaciones Web. REST comprende una serie de limitaciones y principios arquitectónicos; en caso de que una aplicación o diseño cumpla con ellas, es considerada RESTful.

Uno de los características clave de REST en aplicaciones Web, es que la interacción entre el cliente y el servidor no mantiene estado alguno entre solicitudes, y por tanto, toda solicitud del cliente al servidor deberá contener toda la información necesaria para llevar a cabo la solicitud (el cliente no es consciente de si el servidor debe reiniciarse en momento alguno entre las solicitudes).

En el lado del servidor, el estado y la funcionalidad de la aplicación se definen a través de recursos. Un recurso es una identidad conceptual que se expone a los clientes, por ejemplo: objetos de aplicaciones, registros de bases de datos, etc. Cada

recurso tiene un punto de acceso único, definido mediante una URI (Universal Resource Identifier – identificador de recursos universal).

Todos los recursos definidos comparten una interfaz para la transferencia de estados entre cliente (o clientes) y servidor. Se usan métodos estándar HTTP como GET, PUT, POST y DELETE.

Las principios arquitectónicas en los que se basa REST, aplicados como un todo, generan una aplicación que podrá ser escalada sin dificultad a grandes cantidades de clientes. Puede decirse que REST simplifica la implementación, tanto en el lado del cliente como en del servidor.

4.2.1.2 Resto vs SOAP

Desde el origen de la *world wide web* hasta hace relativamente poco tiempo (mediados de la década pasada), los servicios web basados en SOAP construidos con arquitectura de estilo RPC suponían el enfoque más popular a la hora de implementar una Arquitectura orientada a servicios (SOA), en la cual los clientes de un servicio web de estilo RPC envían un *sobre* con datos (información de métodos y argumentos al servidor), utilizando el protocolo HTTP.

El servidor recibe e interpreta el sobre, ejecutando los métodos requeridos (utilizando los argumentos definidos en la llamada). Los resultados del método se empaquetan en un sobre y son devueltos al cliente en forma de respuesta. Al recibir la respuesta, el cliente abre el sobre y actúa en consecuencia, según su propia definición/implementación. Cada objeto tiene sus métodos particulares (que son únicos) y el servicio web RPC expone una sola una URI, que representa el único punto final. Se ignoran la gran mayoría de las características HTTP, usándose sólo el método POST.

El enfoque RESTful para servicios web surge como una alternativa bastante aceptada dada su naturaleza liviana y su capacidad de transmitir datos directamente sobre HTTP. Los clientes se pueden implementar usando una gran variedad de lenguajes (Java,, Ruby, Python, PHP y Javascript (incluyendo Ajax)). Se suele acceder a los servicios web RESTful a través de un cliente automatizado o incluso mediante el uso de una aplicación que actúa en representación del usuario, pero, dada la simplicidad de estos servicios, hay lugar para la interacción humana directa mediante una petición GET a una URL y posterior lectura de la respuesta a través del propio navegador.

En un servicio web de estilo REST, cada recurso tiene una dirección única. Los recursos en sí son los objetivos de las llamadas de los métodos y todos los recursos comparten una misma lista de métodos. Los métodos son estándar y se basan en el estándar HTTP: GET, POST, PUT, DELETE, a lo que pueden añadirse los métodos HEADER y OPTIONS.

En una arquitectura de estilo RPC, el punto clave son los métodos, quedando este lugar reservado a los recursos en toda arquitectura tipo REST, el foco está dado

en los recursos. Las representaciones de los recursos se interconectan mediante hipervínculos que se ubican dentro de la representación.

4.2.1.3 Gestor de recursos: Jersey

Han surgido varios frameworks Java que facilitan la construcción de servicios web RESTful. Restlet es ligero e implementa conceptos tales como recursos, representación, conector y tipo de medio para todo tipo de sistemas RESTful, inclusive servicios web. En el framework Restlet, tanto el cliente como el servidor son componentes. Los componentes se comunican entre sí a través de conectores.

La JSR-311 es una especificación de Sun Microsystems que define un conjunto de APIs Java para el desarrollo de servicios web RESTful. La implementación de referencia de JSR-311 se denomina Jersey.

La JSR-311 proporciona una serie de anotaciones con clases asociadas e interfaces que pueden usarse para exponer objetos Java como recursos Web. La especificación supone que el protocolo de redes subyacente es HTTP. En esta se brindan mapeos claros entre la URI y sus correspondientes clases de recursos, así como también mapeos de métodos HTTP con los métodos de los objetos Java, mediante el uso de anotaciones. La API soporta una amplia gama de tipos de contenidos de entidad HTTP como HTML, XML, JSON, GIF y JPG, entre otros. También se proporciona la capacidad de complementación que permite la inclusión de otros tipos de contenidos en aplicaciones de manera estándar.

Esta especificación es la utilizada en la implementación del proyecto que nos ocupa.

4.2.1.4 Lógica de negocio

Desde el punto de vista de la arquitectura y/o diseño de la aplicación, cabe destacar que para la implementación de la lógica de negocio se ha decidido utilizar la versión estándar del entorno de desarrollo Java, sin necesidad de usar ningún framework específico. El motivo principal es que, dado que la aplicación tiene como requisito no depender de ninguna instalación de base de datos, quedando este aspecto reducido estrictamente a su almacenamiento en un sistema de ficheros en el propio servidor web, hubiera resultado más costoso desde el punto de vista del tiempo (curva de aprendizaje, complejidad en la configuración) y el rendimiento el hecho de utilizar alguno de los frameworks que se tuvieron en cuenta en la fase previa de análisis, como SpringMVC o JSF.

Cabe destacar que la lógica de negocio resultante puede ser considerada bastante ligera desde el punto de vista computacional, puesto que la práctica totalidad de la ejecución caería en la sub capa de acceso a datos, como se comenta en el siguiente punto.

La funcionalidad más “pesada” en esta capa es aquella que se encarga de enviar notificaciones por correo electrónico, que utiliza el API JavaMail.

La implementación de la capa de negocio será detallada en el capítulo siguiente.

4.2.1.5 Acceso a datos

Nos referimos aquí a aquellas operaciones que se efectúan sobre los datos que se reciben o las consultas a la estructura de almacenamiento de datos que se realizan; su objetivo es preparar los datos para que la vista solo se tenga que preocupar de su representación visual, de modo que, en caso de necesidad de cambios en la estructura de almacenamiento o en las operaciones que se realizan con los datos, estos tendrían que ser implementados solamente en el modelo, evitando que un cambio se tenga que realizar en todas las vistas que utilicen unos determinados datos.

Desde el momento en el que opta por el uso de ficheros XML como almacenamiento de datos, y dado que la aplicación va a ser desplegada sobre un servidor de aplicaciones Apache (íntimamente relacionado a la tecnología Java), se decide utilizar JDOM, una biblioteca de código abierto para manipulaciones de datos XML optimizados para Java. Da soporte a acciones tales como parseo, búsquedas, modificación, generación y serialización.

En JDOM encontramos los siguientes paquetes:

- Paquete org.jdom, en el que destacan las clases:
  - Document, que representa el documento XML.
  - Element, que representa el elemento o etiqueta que forma el documento.
  - Attribute, que representa los atributos que puedan tener los elementos.
- Paquete org.jdom.adapters albergará todas las clases adaptadoras.
- Paquete org.jdom.input, que incluye las clases *builder* para construir los documentos XML.
- Paquete org.jdom.output, en el que se encuentran las clases que utilizaremos para dar salida a nuestra clase Document.

Para la lectura de los ficheros XML, una sencilla combinación de la clase File (disponible en la distribución estándar de Java) y los paquetes JDOM anteriormente mencionados cubre todas las necesidades de la aplicación, como se refleja en el siguiente ejemplo:

```
SAXBuilder builder = new SAXBuilder();
File xmlFile = new File(name);
Document document=new Document();
```

La escritura de ficheros se lleva a cabo mediante las clases FileWriter (Java estándar) y XMLOutputter, perteneciente a la librería JDOM:

```
FileWriter fw = new FileWriter(title);
```

```
XMLOutputter serializer = new
XMLOutputter(Format.getPrettyFormat());
serializer.output(content, fw);
```

4.2.2 Capa de presentación

4.2.2.1 Javascript

JavaScript es un lenguaje interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C.

Al contrario que Java, JavaScript no es un lenguaje orientado a objetos propiamente dicho, ya que no dispone de Herencia, es más bien un lenguaje basado en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad.

Todos los navegadores interpretan el código JavaScript integrado dentro de las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del DOM.

El lenguaje fue desarrollado por primera vez en la empresa Netscape Communications, que es la que fabricó los primeros navegadores web comerciales. Apareció por primera vez en el producto de Netscape llamado Netscape Navigator 2.0.

Tradicionalmente, se venía utilizando en páginas web HTML, para realizar tareas y operaciones en el marco de la aplicación únicamente cliente, sin acceso a funciones del servidor. JavaScript se ejecuta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

4.2.2.2 Jquery

El mejor resumen de lo que es jQuery lo podemos encontrar su propio lema: "*La librería JavaScript para escribir menos y hacer más*". Y es que, efectivamente, jQuery es una librería que simplifica la manera de interactuar con documentos HTML, manipular el árbol DOM, manejar eventos e incluso desarrollar animaciones (algo que hasta la aparición de esta librería estaba reservado casi exclusivamente a Flash). Además, permite agregar interacción con páginas web a través de AJAX, que como veremos en el siguiente apartado resulta clave a la hora de implementar este proyecto.

Podemos por tanto considerar que jQuery es un Framework de Javascript. Es decir, un conjunto de funciones que ya fueron desarrolladas y probadas, y están listas para ser utilizadas de una manera muy simplificada.

jQuery consiste en un único fichero JavaScript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX, que ha de ser importado en toda página en la que cualquiera de sus funcionalidades vaya a ser referenciada.

4.2.2.3 AJAX

AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript y XML asíncronos, donde XML es un acrónimo de eXtensible Markup Language), es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

De esta manera, resultaría interesante valernos tanto de AJAX como de jQuery, dadas sus características; y es algo perfectamente posible, puesto que gracias al conjunto de métodos que posee es posible llevar a cabo todo el proceso de petición al servidor Web, y procesar de la respuesta en unas pocas líneas de código. Esto se consigue con el método jQuery.ajax(), ya que con una sola llamada a esta función podemos configurar todo el proceso de la petición AJAX. La documentación detallada relativa a jQuery y AJAX se puede encontrar en la propia referencia de jQuery (http://api.jquery.com/category/ajax/), pero para este proyecto, valga destacar que la relación AJAX-jQuery nos proporciona una manera ideal de implementar la comunicación cliente-servidor.

4.2.3 Capa de almacenamiento de datos

4.2.3.1 Esquema de relación datos-entidades

En el capítulo 3 se procedió a describir el conjunto de relaciones y entidades envueltas en la concepción de la idea general de este proyecto. La implementación de cada una de ellas así como todas las relaciones derivadas dan lugar al modelo de datos que se detalla en la siguiente gráfica:

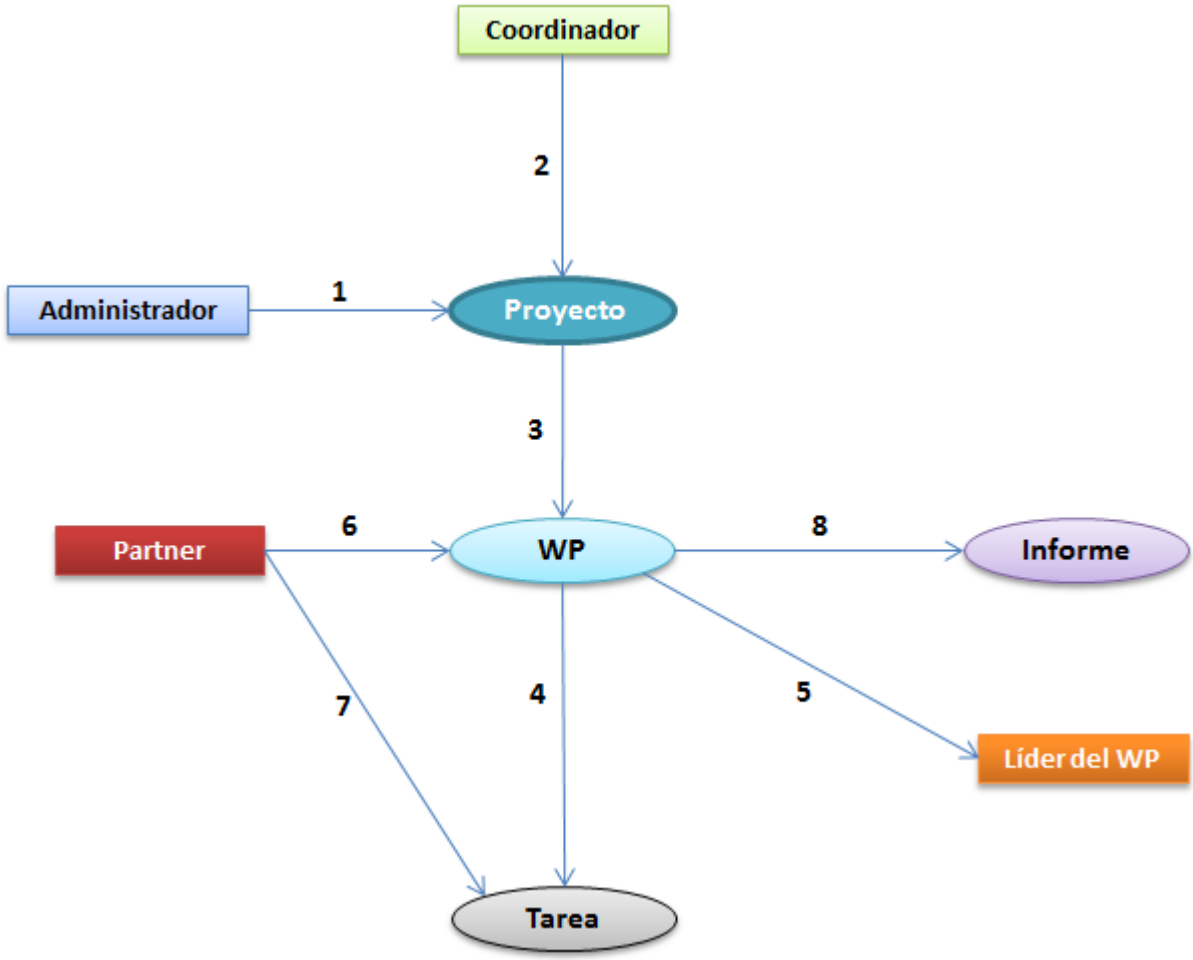


Figura 15: Relación datos-entidades

#	Descripción de la acción
1	El administrador crea un proyecto, asignando un coordinador
2	El coordinador gestiona el proyecto, añadiendo WPs y tareas y asignando partners a cada uno de ellos
3	Un proyecto puede contener uno o más WPs
4	Un WP puede contener una o más tareas
5	Cada WP tiene asignado un partner que actúa como líder
6	Cada WP tiene asignados uno o varios partners que trabaja en él
7	Cada tarea tiene asignada uno o varios partners que trabaja en ella
8	Cada WP tiene definidos distintos sub informes; uno por cada conjunto de partner y fecha de entrega de informe

Desde un punto de vista más estricto, ciñéndonos a un modelo entidad-relación más puro, tenemos:

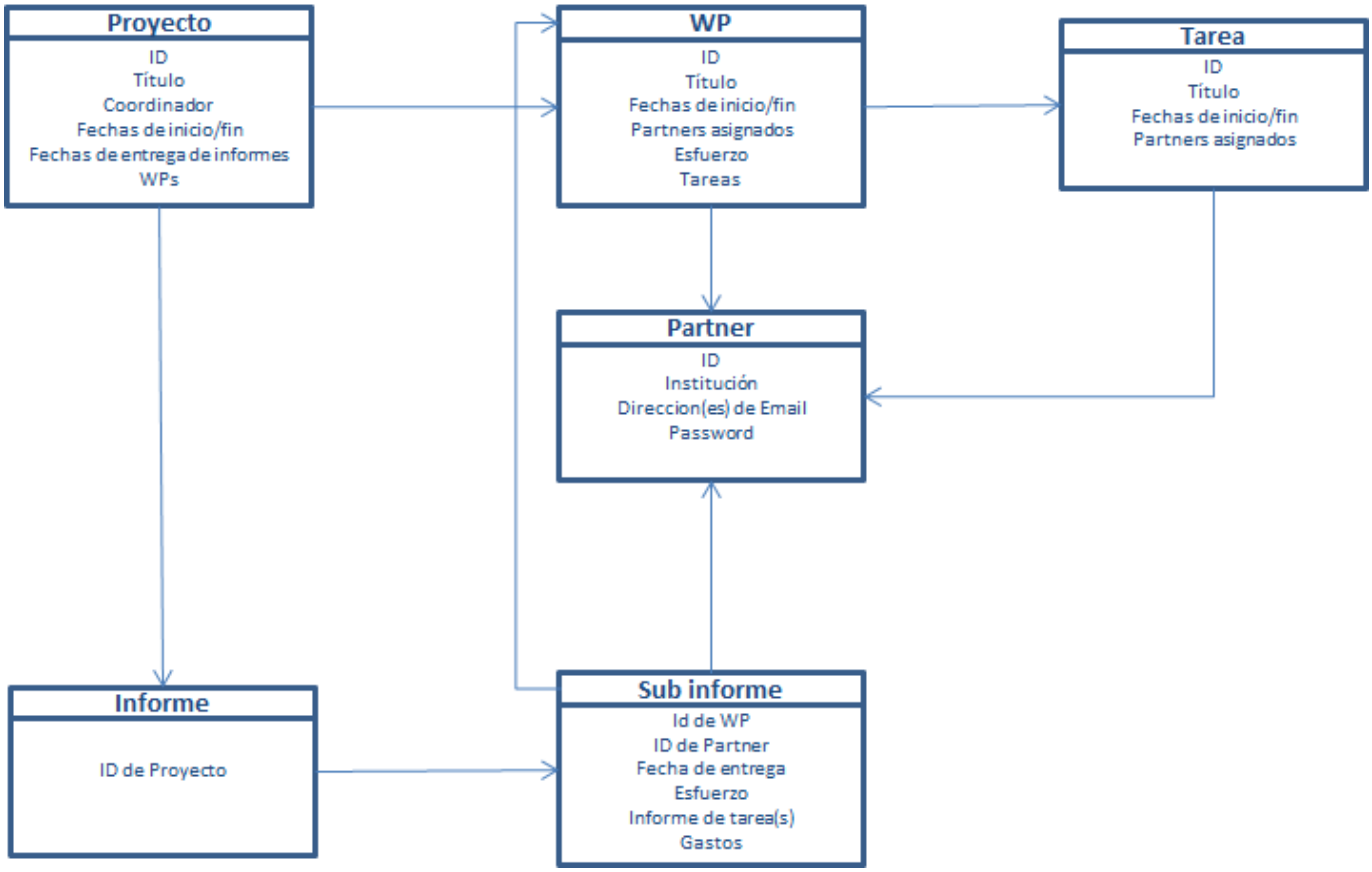


Figura 16: Propuesta de modelo entidad-relación

4.2.3.2 Almacenamiento de datos

La implementación física de este modelo ha de tener en cuenta uno de los requerimientos principales definidos en este proyecto: la portabilidad de todos los potenciales contenedores de información y datos, para la que desde las primeras conversaciones con el patrocinador del proyecto se acordó utilizar ficheros XML (del inglés eXtensible Markup Language).

XML es un sistema estándar de codificación de información. Los programas que utilizan el formato XML pueden intercambiar fácilmente sus datos, ya que responden a una misma lógica interna. Los documentos XML son ficheros de texto que contienen la información organizada en forma de árbol: cada rama puede tener unos atributos propios y servir de base para otras ramas. Además, los documentos XML se pueden transformar o combinar: un 'tronco' (elemento padre) con todas sus 'ramas' (hijos) puede pasar a ser una rama de otro 'árbol mayor'.

Un ejemplo de documento XML abreviado podría ser:

```
<comunidad nombre="Extremadura">
        <provincia>Cáceres</provincia>
        <provincia>Badajoz</provincia>
</comunidad>
```

No fue esta una decisión arbitraria, sino que se tuvieron en cuenta distintos factores antes de elegir esta opción como la más adecuada.

En primer lugar, la flexibilidad en la estructura de este tipo de ficheros hace que nuevas características (atributos) puedan ser añadidas a cualquiera de las entidades sin mayor repercusión en la lógica de negocio existente.

Del mismo modo, las previsiones de uso de esta herramienta no contemplan la posibilidad de múltiples accesos simultáneos, lo que de haber sido podría dar lugar a pensar en algún tipo de mecanismo de sincronización a nivel transaccional; pero es que, además, hay que tener en cuenta que el diseño de esta aplicación RESTFul, en el que no se implementa mantenimiento de sesión alguna entre llamadas, hace que el ámbito de toda transacción se reduzca a un período mínimo, que en las distintas pruebas realizadas nunca ha superado los 3.8 segundos (tiempo que va desde que se hace la llamada REST hasta que la lógica de negocio es ejecutada), de modo que la opción ficheros XML sigue manteniéndose como la más óptima para este desarrollo.

4.2.4 Arquitectura multinivel para la construcción de servicios web RESTful

Una vez descritos todos los componentes existentes en la implementación técnica de esta aplicación, resulta lógico utilizar el concepto de Arquitectura Multinivel para referirnos al diseño finalmente escogido (el patrón MVC es una implementación de esta arquitectura). Los servicios web RESTful y las aplicaciones Web dinámicas tienen bastantes puntos en común, como el hecho de que proporcionan datos y funciones similares, aunque para distintos tipos de clientes. Los servicios web pueden llegar a beneficiarse con la separación de responsabilidades propia de una arquitectura multinivel del mismo modo que lo hacen las aplicaciones Web más dinámicas. Tanto datos como lógica de negocios pueden compartirse entre clientes tanto automatizados como interfaz de usuario. Las únicas diferencias vienen dadas por el tipo de cliente y la capa de presentación (vista). Además, la separación entre la lógica de negocios y el acceso a los datos posibilita la independencia de las bases de datos o cualquier otro tipo de almacenamiento, tal como los ficheros XML utilizados en esta aplicación.

La figura 17 muestra la arquitectura multinivel definida para la implementación de este proyecto; en posteriores epígrafes se propondrá una solución para una futura expansión de las funcionalidades de esta herramienta, utilizando otros clientes distintos al navegador web, como pueden ser dispositivos móviles y/o tabletas.

Cabe destacar también la existencia de los denominados clientes automatizados, como pueden ser Java, scripts de lenguajes como Python, Ruby o PHP y herramientas de la línea de comandos como Curl. También pertenecen a este grupo los Ajax, GWT, o blogs que se ejecutan dentro del navegador y actúan como consumidores del servicio web RESTful, debido a que lo hacen de manera automatizada y en representación del usuario. Los clientes de servicios web envían solicitudes HTTP al Resource Request Handler en el nivel Web. Las solicitudes sin estado de los clientes contienen la información de métodos en el encabezamiento: POST, GET, PUT y DELETE, que serán mapeados hacia las operaciones correspondientes de los recursos que se encuentran en el Resource Request Handler. Cada solicitud contiene toda la información necesaria, incluso las credenciales que permiten al gestor de recursos (Resource Request Handler) procesar la solicitud.

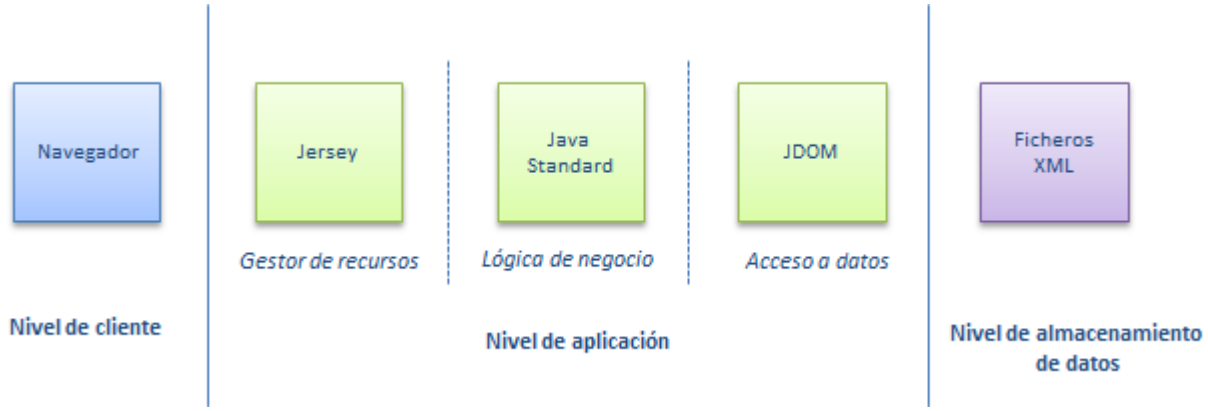


Figura 17: Diagrama de un entorno de aplicación Web de varios niveles

Cuando el componente RRH recibe una solicitud del cliente (aplicación web en este caso), éste solicita a su vez el servicio a la siguiente capa (lógica de negocio). El Resource Request Handler identifica todas las entidades que el sistema expone como recursos y asigna una URI única a cada una de ellas. El componente RRH deberá ser ligero y delegar el trabajo pesado principalmente al nivel de negocios.

Por definición, Ajax y los servicios RESTful se ajustan uno al otro, aprovechando tecnologías Web de fácil disponibilidad y estándares como HTML, JavaScript, objetos de navegador, XML y/o JSON y HTTP. Por tanto no es necesario utilizar ningún otro componente externo para lograr una interacción óptima entre el frontend Ajax y los servicios RESTful.

La implementación de los requerimientos propios de la aplicación se localiza en la capa de lógica de negocio, que actúa como intermediario en los intercambios de datos entre las capas de presentación y datos.

La capa de datos proporciona el nivel de almacenamiento de datos a la interfaz. En el proyecto que nos atañe, se ha optado por un modelo no estándar que se basa en objetos JDOM, resultado de las operaciones de lectura y escritura sobre ficheros XML, que actúan como medio de almacenamiento para la información de la aplicación, tanto a nivel de configuración como de datos de usuario. Esta decisión viene determinada por el requerimiento de tener un almacenamiento flexible y desacoplado, que permita migrar la herramienta sin ningún tipo de dependencia con Base de Datos y sus consiguientes instalaciones y tareas de mantenimiento.

4.2.5 Conclusión

REST define un estilo arquitectónico de sistemas en red como, por ejemplo y ciñéndonos a nuestro caso, aplicaciones Web. Las especificaciones REST, aplicadas como un todo, dan lugar a una arquitectura simple, escalable, y eficiente. Los servicios web RESTful han surgido como una alternativa distinta a los anteriores estándares basados en servicios SOAP por su naturaleza más simple y ligera, además de por su capacidad de transmitir datos directamente sobre HTTP. La arquitectura multinivel tanto para servicios web como para aplicaciones Web dinámicas nos lleva a la reutilización, simpleza, extensibilidad y a una clara separación de las responsabilidades de los componentes. Ajax y los servicios web RESTful se ajustan naturalmente el uno al otro. Usando Ajax y los servicios web RESTful de forma conjunta, los desarrolladores logran crear interfaces de alta calidad.

Analizando estas características y sus evidentes ventajas para la implementación de este proyecto, queda abierta la posibilidad futura de modificar cualquiera de los componentes (modelo, vista, controlador) de forma individual sin que ello afecte al resto.

5 IMPLEMENTACIÓN

5.1 Aplicación Web

5.1.1 Backend

Tomando como referencia el apartado dedicado al diseño y arquitectura de la aplicación, nos centramos ahora en describir su implementación técnica. Se ha creado una librería Java que contiene todas las clases necesarias para cubrir las funcionalidades requeridas, dividiéndose en tres paquetes según el tipo de funcionalidad.

5.1.1.1 Controlador (adaptador de recursos): paquete com.reportingtool.rest

Implementación del controlador Jersey, que decide qué recurso ejecutar basándose en la URL solicitada. Dicho controlador ha sido implementado en una clase (Dispatcher.java) en la que se exponen todos los recursos.

Un ejemplo de configuración de un recurso a través de Jersey podría ser:

```
@Path("/rest/test")
public class RestService {

   @GET
   @Path("/hello")
   @Consumes(MediaType.APPLICATION_XML)
   @Produces(MediaType.TEXT_PLAIN)
   public String HelloWorld() {
      //Lógica a implementar
      return "Hello World";
   }
}
```

− La primera línea define la URL base a partir de la cual las peticiones serán manejadas por esta clase; de esta manera, una llamada a {URL dominio}/rest/test/{recurso}, será 'interceptada' y manejada en esta clase, mientras que una llamada distinta, a por ejemplo {URL dominio}/rest/prod/{recurso} no será tenida en cuenta, y deberá buscar por otro adaptador que se haga cargo de ella.

− @GET: Define qué método HTTP es el que se corresponde con la ejecución de esta función; si se recibe una petición mediante un método distinto a GET (POST, PUT o DELETE), se lanzará una excepción. Si no se define método alguno, cualquiera de ellos será válido y la función se ejecutará normalmente.

- @Path("/hello"): Identificador del recurso.
- @Consumes/Produces: Especifica qué tipo de datos espera la aplicación y qué tipo será generado en la salida.

Una vez visto este ejemplo, se listan a continuación todos los recursos definidos en la aplicación. Estas URLs definen el API REST de esta aplicación, cuyo detalle de adjunto en el Anexo I.

/report/get/{projectid}
/report/addexpenses/{currentProject}/{currentWP}/{currentPartner}/{currentReport}
/report/edit/{currentProject}/{currentWP}/{currentPartner}/{currentReport}
/report/task/edit/{currentProject}/{currentWP}/{currentPartner}/{currentReport}/{taskId}
/report/sendbyemail/{currentProject}/{currentWP}/{currentPartner}/{currentReport}
/report/edit/{currentProject}/{currentWP}/{currentPartner}/{currentReport}
/partners
/partners/add
/partners/login
/partners/forgot
/projects
/project/{projectid}
/project/create
/project/add/schedule/{projectIdinUse}
/project/add/partner/wp/{projectIdinUse}/{wpId}
/project/add/partner/task/{projectIdinUse}/{wpId}/{taskId}
/project/add/WP/{projectIdinUse}
/project/add/task/{projectIdinUse}/{wpId}
/project/{projectId}

Tabla 3: Lista de recursos disponibles

5.1.1.2 Lógica de negocio de las entidades: paquete com.reportingtool.entities

Dentro de este paquete, se encuentra la lógica de negocio relacionada con cada entidad, con una clase Java por cada una de ellas: Proyecto (Project.java), Partner (Partner.java), Informe (Report.java). A continuación se listan las más destacadas funciones disponibles para cada una de ellas:

Project.java:
addProject: Crear nuevo proyecto.
addSchedule: Añadir fechas de entrega de informes.
addWP: Añadir nuevo WP.
addTask: Añadir nueva tarea.
assignPartnersToTask: Asignar partners a una tarea.
assignPartnerToWP: Asignar partner a un WP.
createReportStructure: Crear el fichero con la estructura del informe ({projectid}_report.xml).
getCurrentProjectDocument: Obtener el fichero de proyecto actual.
getProjects: Obtener todos los proyectos definidos en el fichero de Proyectos.

Partner.java

addPartner: Añade nuevo partner.
getCurrentPartnersFile: Obtiene el fichero de partners actual.
removePartner: Elimina un determinado partner.

Report.java
addExpenses: Añade una nueva entrada de gastos.
addSubReport: Crea un nuevo subinforme.
addTaskReport: Añade un nuevo informe de trabajo para una determinada tarea.
fillReportFile: Rellena el fichero de informe de un proyecto.
getCurrentReportFile: Obtiene el fichero de informes correspondiente.
sendReportByEmail: Envía un subinforme por email.
updateSubReport: Actualiza un determinado subinforme.
updateTaskReport: Actualiza un informe de trabajo para una tarea.

5.1.1.3 Utilidades: paquete com.reportingtool.utils

Contiene funciones comunes, que pueden ser usadas por todas las entidades indistintamente, destacando getDate() para obtener la fecha actual, docToString() para convertir un Documento JDOM en texto plano o sendEmail() para enviar cualquier tipo de información por correo electrónico.

5.1.2 Frontend

5.1.2.1 Página HTML

Una de las características más destacadas de esta aplicación web es que toda su ejecución ocurre dentro de una misma página (app.html). Evidentemente, cada interacción del usuario con cada uno de los componentes que hacen uso de alguno de los servicios publicados requiere de comunicación con el servidor, pero todas ellas están encapsuladas a nivel Javascript, con lo que no se produce cambio alguno de página durante la estancia del usuario en la aplicación.

Esta página está diseñada siguiendo las recomendaciones HTML5 para la separación y muestra de contenido; si bien se trata única y exclusivamente de una recomendación y no de una condición exabrupto a la hora de desarrollar sitios web, es cierto que en el caso de la aplicación que nos ocupa satisface ampliamente las necesidades definidas; el hecho de utilizar artículo para presentar la información nos permite agrupar en cada uno de ellos la representación de datos relevantes para un determinado epígrafe.

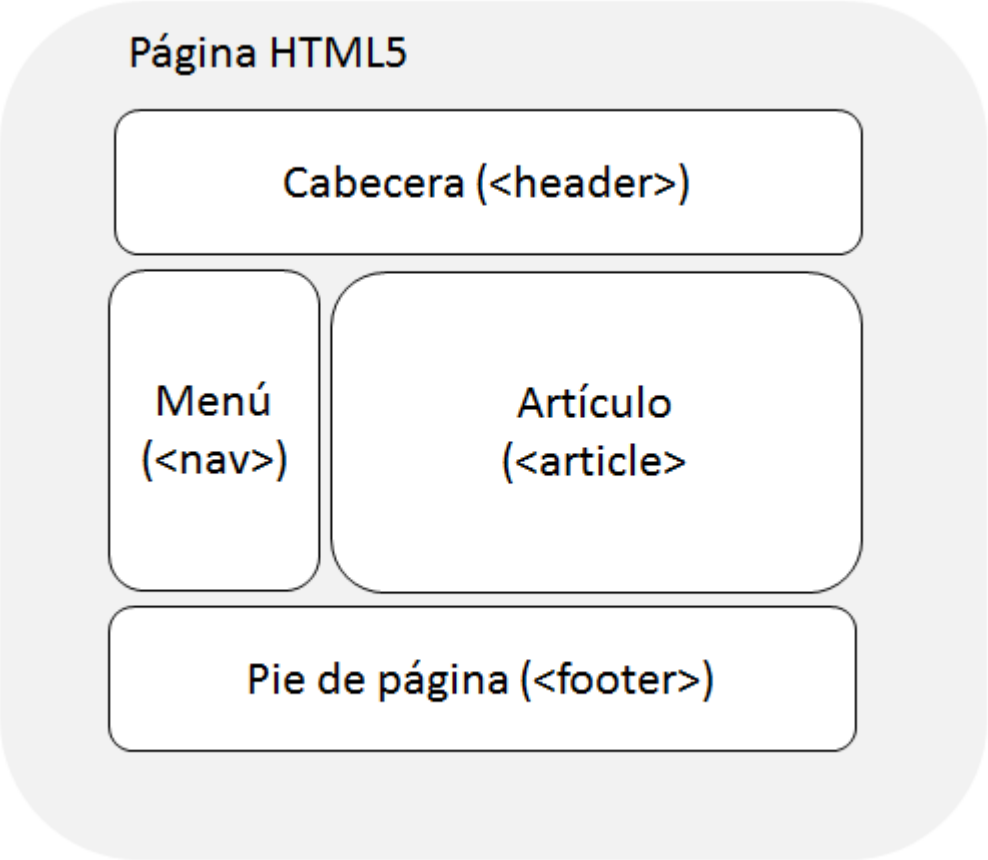


Figura 18: Estructura de una página HTML5

Y es que, gracias a Jquery, se pueden mostrar/ocultar los artículos de manera sencilla, lo que nos permite llevar a cabo una completa navegación a lo largo y ancho de la aplicación sin dejar la página de entrada, como se explicaba anteriormente.

Cabe destacar que gran parte del código HTML está definido en la página de forma estática, si bien otra considerable parte es generada mediante Jquery y añadida a los nodos correspondientes (elementos HTML que contienen un identificador único). Un pequeño ejemplo que ilustra esta funcionalidad (ampliamente usada en la herramienta) sería:

$('#selectWpAddTask').append('<option value="'+wpId+'">'+wpTitle+'</option>');

En este ejemplo, se añade a un determinado elemento definido con identificador "selectWpAddTask" (en este caso, un selector de valores para un formulario) una línea de código HTML que añade un nuevo valor a dicho selector.

5.1.2.1.1 Interfaz de Proyectos

- Componentes: Menú de navegación, Artículo contenedor de secciones, cabecera con menú.

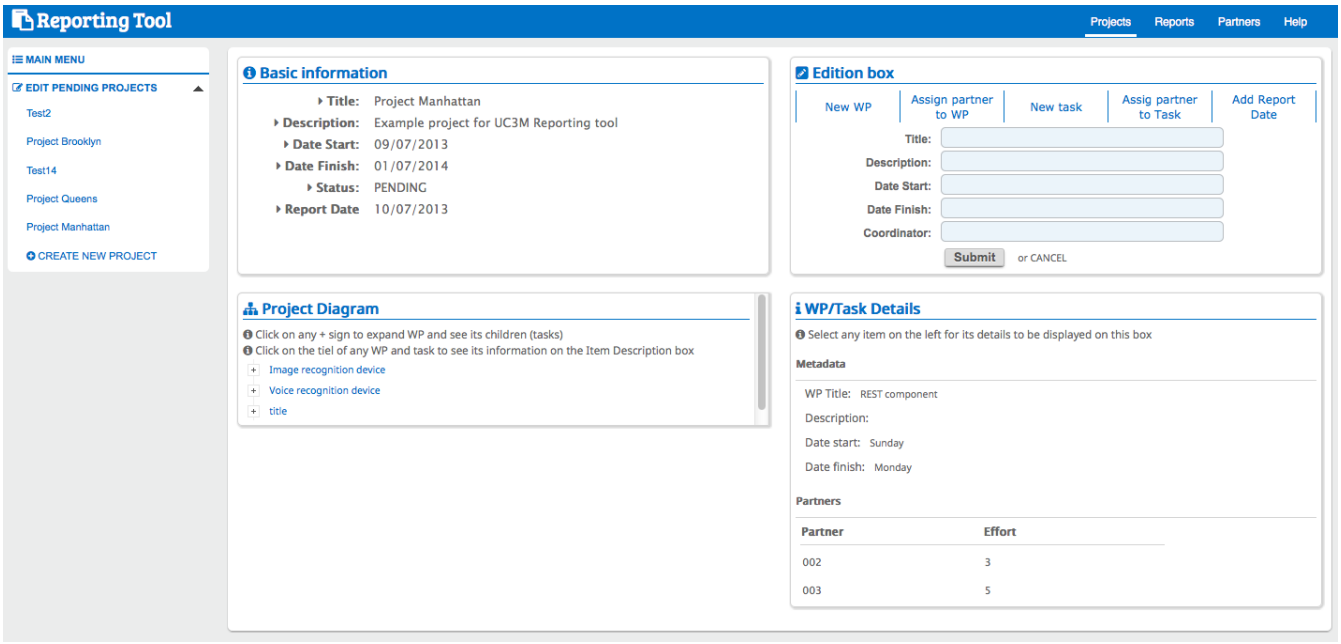


Figura 19: Vista de proyectos; Estructura de página

5.1.2.1.2 Interfaz de Informes: Lista de subinformes

- Componentes: Menú de navegación, Artículo contenedor de tabla de listado de subinformes, cabecera con menú.

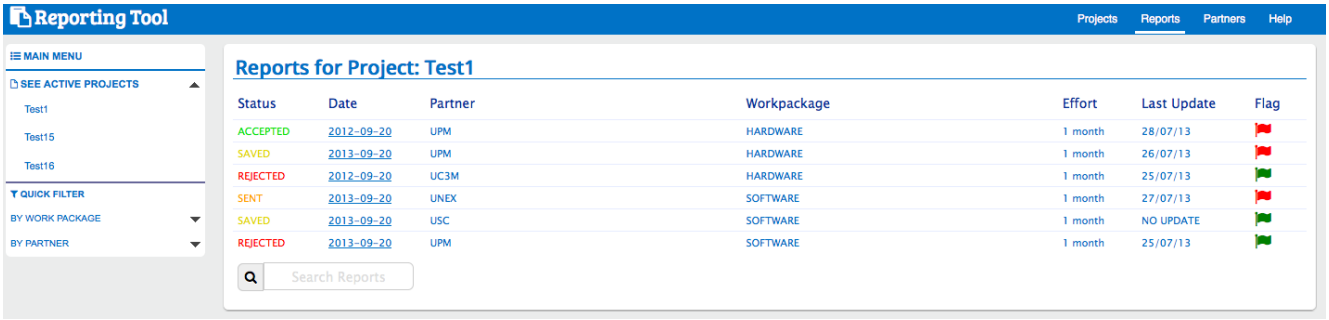


Figura 20: Listado de subinformes; elementos de la página

5.1.2.1.3 Interfaz de informes: Detalles de un subinfome

- Componentes: Menú de navegación (*), Artículo contenedor de secciones, cabecera con menú (*).

(*) Para una mejor visualización, se ha recortado la captura de pantalla, ocultando estos elementos.

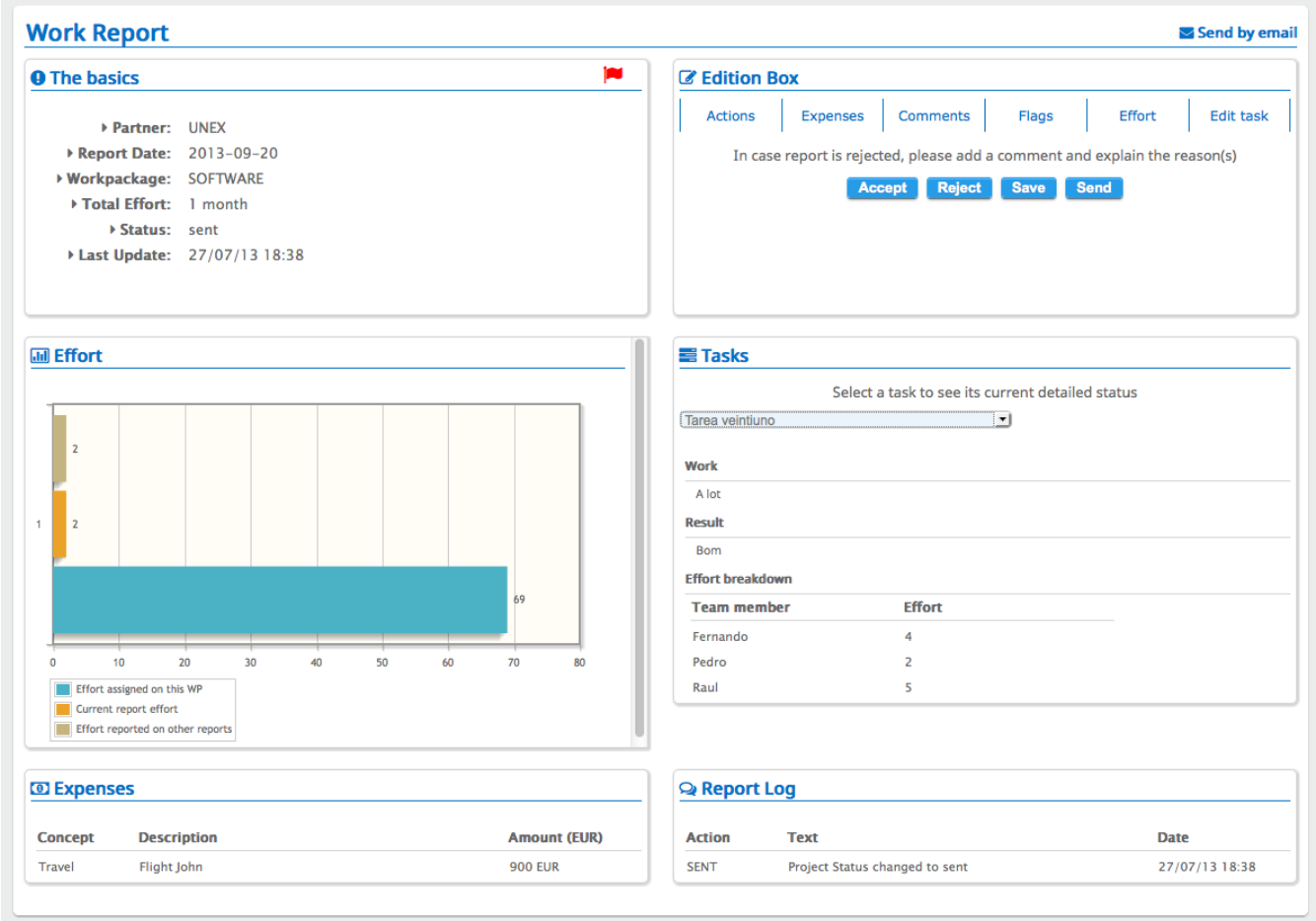


Figura 21: Vista de subinformes; elementos de la página

Como se puede comprobar, se ha respetado la recomendación HTML5 para la estructura de la página. La cabecera es común a todas las vistas, y el contenido del menú de navegación es dinámicamente modificado en función de la vista en uso.

En cuanto a la parte central de la página, la implementada a partir de artículos, podemos destacar que consta de una serie de secciones (Basic Information, Edition Box…). Se trata simplemente de contenedores HTML (elementos tipo “section”) que sirven para separar físicamente los distintos tipos de datos a mostrar. En el caso de las secciones de edición (Edition Box), además encontramos un menú que permite mostrar distintos formularios a través de los cuales el usuario puede interactuar con la herramienta, bien sea añadiendo elementos al proyecto o redactando un determinado subinforme de trabajo.

5.1.2.2 Librerías Javascript propias

Con el objetivo de ordenar el código Javascript, se ha decidido crear cuatro ficheros distintos; en dicha organización, se han tenido cuenta aspectos funcionales, cubriendo cada fichero las acciones definidas para cada uno de los principales perfiles de la aplicación: edición de proyectos, edición de informes y gestión de usuarios.

project.js
Funciones principales: updateProjectView, getProject
Descripción: Implementa todas las funciones necesarias para mostrar la información de un proyecto (parseando el resultado de la llamada al API REST que devuelve la información del mismo) así como aquellas que se encargan de enviar las interacciones del usuario con los formularios que permiten modificar la estructura del proyecto.

Contiene todas las funciones relacionadas con la creación y edición de proyectos (descritas en el apartado 3.4, Casos de uso). En él, se implementan las llamadas AJAX que se encargan de crear un proyecto, añadir WPs, añadir tareas, asignar partners…

Podemos distinguir dos aspectos fundamentales dentro de este fichero: los elementos encargados de mostrar la información del proyecto y los que sirven como punto de entrada de los datos que modifican dicha información del proyecto.

Para el primer caso, la implementación consiste en realizar una petición HTTP para obtener la representación del fichero XML en el que se define el proyecto (función getProject(projectId)); una vez obtenida la respuesta, se utiliza Jquery para recorrer dicho fichero, obteniendo la información de cada uno de sus nodos y presentándosela al usuario a través de una serie de apartados (diagrama de proyecto, información básica…) definidos en la estructura HTML (ver Figura 21). Para ello, nos valemos del anteriormente mencionado método .append(), añadiendo información dinámica a los nodos HTML.

En el caso de los formularios que sirven como puntos de entrada de datos del usuario, nos valemos de selectores Jquery para manejar dichos formularios; ellos procesan los datos introducidos y los incluyen en la correspondiente llamada al API REST, resultado de la cual el fichero será modificado con la información incluida en la llamada.

La otra función clave es updateProjectView(projectId), la cual se encarga de refrescar la vista mediante una llamada a getProject() cada vez que se produce algún evento que modifique la estructura de datos.

Para ilustrar esta explicación con un ejemplo, pensemos en el momento en el que el coordinador del proyecto crea un nuevo WP:

1. Antes de nada, al seleccionar el proyecto en cuestión, se produce una llamada AJAX al API REST para recuperar la información del proyecto
2. La respuesta es parseada a través de Jquery
3. El coordinador rellena el formulario de creación de WP, y al enviar dicho formulario, otra llamada AJAX al API REST provocará que el fichero XML en el servidor sea modificado, añadiéndose el nuevo WP.
4. Una vez ejecutada finalizada dicha llamada AJAX, se ejecuta la función updateProjectView(), que actualiza la vista con la última versión del fichero XML.

coordinator.js
Funciones principales: showListOfReports,showReport,updateView ▪ **Descripción**: Contiene todas las funciones que se encargan de mostrar al usuario los subinformes de un proyecto, así como el detalle de cada uno de ellos una vez que sean seleccionados. Además, maneja todos los formularios que permiten modificar los valores de un determinado subinforme.

A nivel funcional, comparte las características principales de la librería project.js, ya que utiliza el mismo concepto de mostrar la información de los subinformes de un proyecto así como permitir la entrada de datos a través de formularios.

Cabe destacar que esta librería se encarga de mostrar dos artículos HTML distintos; el primero es la lista de subinformes, en forma de tabla, con la información básica de cada uno de ellos; y por otro lado, el artículo en que se detalla un subinforme en concreto (seleccionado en la lista anterior).

El flujo de trabajo es similar al descrito en el punto anterior:

1. El usuario selecciona un proyecto
2. Se muestran los subinformes para ese proyecto (como resultado de la llamada al API REST para recuperar el fichero de informes de ese proyecto)
3. El usuario selecciona uno de estos subinformes (llamada al API REST con el identificador del proyecto en cuestión y del subinforme seleccionado)
4. Se muestran los detalles de este subinforme, incluyendo los formularios para modificar el estado del informe
5. En caso de que uno de estos formularios sea utilizado, se llama a la función updateView(), que refresca la vista con la última versión del fichero del informe.

De nuevo, toda la implementación se basa en selectores Jquery que acaban realizando llamadas AJAX y en parseos de los ficheros XML obtenidos en estas llamadas AJAX.

Existen otras tres librería Javascript que podríamos denominar secundarias, en el sentido de que las funcionalidades que implementan no son tan extensas como las explicadas en los dos puntos anteriores:

initMenu.js: Conjunto se selectores jQuery que inicializa los menús de navegación
partners.js: Selectores jQuery encargados de gestionar el artículo que muestra la información de los Partners de la herramienta.

5.1.2.3 Librerías Javascript externas

- Jquery

Para tener acceso a todas las funcionalidad que jQuery provee, es necesario importar las librerías jquery.js (implementación estándar) y jquery-ui.js (implementación de aspectos de interfaz de usuario). Para esta aplicación, se ha usado la versión 1.8.2 de ambas.

- jqPlot

La librería utilizada generar gráficos (en este caso, relativas al esfuerzo de los partners) se llama jqPlot. Se trata de un plugin jQuery que permite crear gráficas lineales, de barras y circulares (en forma de tarta). Para utilizar sus funcionalidades, basta con seguir las instrucciones de uso especificadas en los manuales de usuario de la propia librería ([14]), utilizando los valores que necesitemos en cada caso (obtenidos de los datos de nuestros informes).

- SiteMapStyler.js

Esta sencilla librería es la encargada de dar formato al diagrama que generamos para representar un proyecto (estructura jerárquica de WPs y tareas).

5.1.3 Seguridad

Esta aplicación se basa en el login inicial de los usuarios para poder acceder a la misma. Es por ello que se han implementado algunas medidas de seguridad en el dicho proceso de login, como cifrado simétrico y comprobación de marca de tiempo:

- **Cifrado MD5**: Es este un algoritmo de seguridad, que recibe una cadena de caracteres y devuelve un número de 128 bits. Cuando el usuario usa el formulario de login, su contraseña pasa por un módulo de MD5, siendo el número generado el que viaja por la red, de modo que la contraseña “real2 no es visible a posibles intercepciones.

- **Marca de tiempo**: Cuando el usuario hace login, se genera una marca de tiempo mediante código Javascript; el servidor la recibe conjuntamente con el resto de elementos del formulario, y genera otra de la misma manera; al

comparar ambas, sólo se permite el acceso si difieren menos de 5 segundos, para evitar que una posible intercepción de una llamada de login sea replicada.

5.1.4 Ficheros CSS

Las siglas CSS (del inglés Cascading Style Sheets, *Hojas de Estilo en Cascada*), se refieren a un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla.

CSS se utiliza para dar estilo a documentos HTML separando el contenido de la presentación. Los Estilos definen la forma de mostrar los elementos HTML; CSS permite a los desarrolladores Web controlar el estilo y el formato de múltiples páginas y/o elementos web al mismo tiempo. Cualquier cambio en el estilo definido para un elemento en la configuración CSS afectará a todas las páginas CSS en las que aparezca ese elemento.

En esta aplicación, utilizamos dos ficheros CSS distintos. En primer lugar, el propio de la aplicación, el que da formato a todos los elementos creados en exclusiva en este desarrollo: tool.css. Y, por otro lado, el uso de la librería jqPlot hace que para dar formato a los gráficos a través de ella generados sea necesario otro fichero: jquery.jqplot.min.css.

5.1.4.1 Fuentes

Por defecto, cualquier navegador puede mostrar una serie de fuentes de uso común y extendido; aun así, para personalizar el diseño de la aplicación, se ha decidido utilizar una fuente gratuita proporcionada por Google (Noto Sans). Para su utilización, basta con importar el fichero correspondiente (http://fonts.googleapis.com/css?family=Noto+Sans) y utilizar dicha fuente en la configuración CSS.

5.1.4.2 Imágenes

El propio logotipo de la aplicación muestra lo que parecer un icono que recuerda a un informe; pues bien, dicha imagen no se basa un archivo de imagen al uso (.jpg, .gif…). Tanto él como el resto de elementos gráficos (normalmente, representando un epígrafe) que nos encontramos en la herramientas han sido generados a través de un tipo de fuente especial llamado FontAwesome. Mediante el uso de la propiedad font-face de CSS3 y la etiqueta de HTML5, FontAwesome proporciona acceso a más de 300 iconos predefinidos. Para hacer uso de ellos, basta con importar el fichero css correspondiente (font-awesome.min.css) y, valiéndonos del tag de HTML5, mostrarlo por pantalla (cada icono tiene un identificador de clase que determina su representación):

```
<i class="icon-dropbox"></i>
```

A efectos del sistema, funciona como si de una fuente tipográfica se tratara por lo que podemos cambiar su tamaño, su color, etc. fácilmente desde CSS.

5.2 Sistema de ficheros

Como se ha venido explicando a lo largo de la memoria, se ha optado por basar el almacenamiento de datos en ficheros XML; a continuación se muestra la estructura de los ficheros involucrados en la aplicación.

5.2.1 Partners.xml

Contiene la información de cada uno de los partners que han sido dados de alta en la herramienta.

5.2.2 Projects.xml:

En este fichero podemos encontrar una referencia a cada uno de los proyectos que han sido generados a través de la herramienta, incluyendo el estado actual de cada uno de ellos.

5.2.3 {ID del proyecto}.xml:

Al crear un proyecto en la herramienta, se genera un fichero que contendrá toda la información introducida (WPs, tareas, partners, fechas…).

5.2.4 {ID del proyecto}_report.xml

Una vez que el proyecto ha sido definido, el coordinador es el encargado de "cerrar" esa fase de definición, momento en el cual la estructura de este fichero de informe es creada. Estos ficheros (uno por proyecto) contienen todos los subinformes (que, recordemos vienen definidos por la combinación de WP-partner-fecha de entrega) para este proyecto determinado.

5.3 Diagrama de relación entre clases (y librerías)

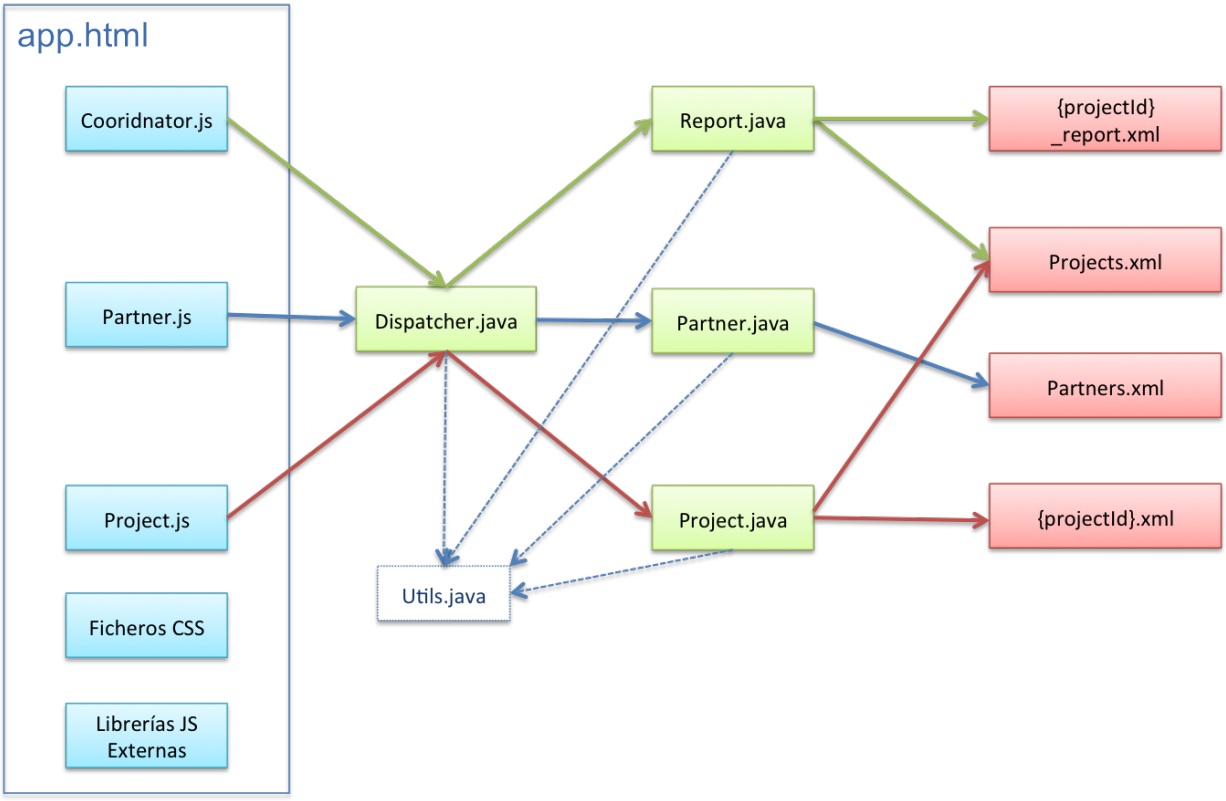


Figura 22: Diagrama de relación entre clases, librerías y ficheros

6 VALIDACIÓN

Se describen en este capítulo los pasos seguidos para instalar y validar el funcionamiento de la aplicación.

6.1 Entorno de desarrollo

6.1.1 Pre requisitos

- Descargar e instalar la última versión de Tomcat
- Descargar las librerías Jersey; es recomendable utilizar el paquete que incluye todas las librerías necesarias (jersey-bundle-x.y.z.jar)
- Descargar las otras librerías Java que se necesitan en el proyecto: jdom-2.0.x y asm-3.1.jar
- Instalar Eclipse (este ha sido el entorno de desarrollo utilizado en esta aplicación y por tanto las instrucciones se basa en él).

6.1.2 Pasos a seguir

- Integración Tomcat-Elicpse
  - Iniciamos Eclipse. Nos dirigimos a Window -> Preferences -> Server -> Runtime Environment, pulsamos Add y seleccionamos nuestra versión de Tomcat. Marcamos *Create a new local server* si no está seleccionado.
  - Pulsamos Next y buscamos el directorio en el que instalamos Tomcat. Pulsamos Finish y OK. El nuevo servidor debería mostrarse en la pestaña Servers. También se habrá creado un proyecto nuevo Servers con los archivos de configuración de nuestra instancia de Tomcat.

- Crear un Proyecto Web dinámico en Eclipse

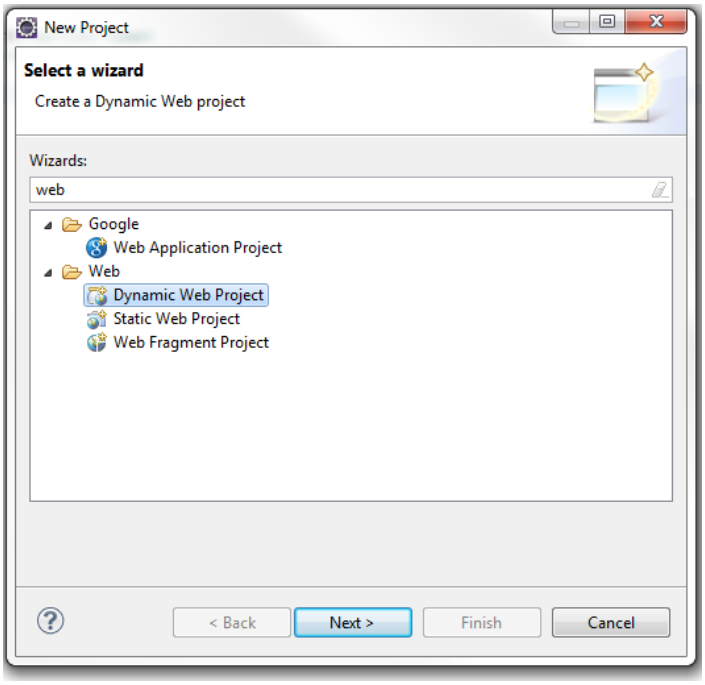


Figura 23: Crear proyecto en Eclipse

- Copiar las librerías contenidas en el paquete de Jersey, así como las librerías Java jdom-2.0.x y asm-3.1.jar, al directorio /lib del proyecto:

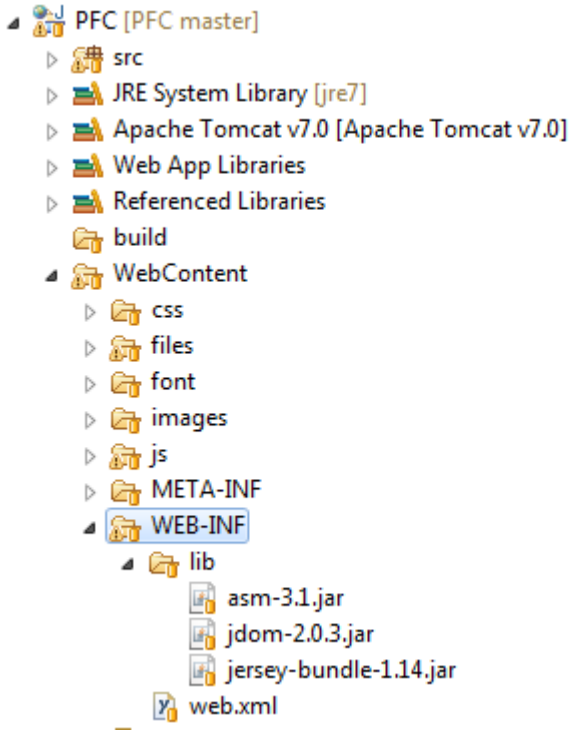


Figura 24: Añadir librerías a proyecto en Eclipse

- Descriptor de despliegue

Es necesario añadir la declaración del servlet Jersey al fichero de configuración de la aplicación web ({Proyecto}/WebContent/WEB-INF/web.xml). Véase el ejemplo concreto de la configuración en nuestra aplicación:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
 <servlet>
  <servlet-name>JerseyTest</servlet-name>
  <servlet-class>
          com.sun.jersey.spi.container.servlet.ServletContainer
       </servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>JerseyTest</servlet-name>
  <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
</web-app>
```

- Gestor de recursos

Según lo indicado en el apartado 5.1.1.1, hemos de crear una clase o clases que se encarguen de implementar todas las funcionalidades que se pretendan asociar a los recursos en cuestión. Es conveniente empezar con una clase de prueba, que contenga un recurso que simplemente devuelva un mensaje, y a partir de aquí empezar a trabajar en recursos más complejos; el mencionado ejemplo del apartado 5.1.1 es perfectamente válido para este caso.

- Prueba

Acceder al recurso a través de http://localhost:8080/{dirección del recurso}. Si seguimos el ejemplo del apartado 5.1.1, la URL sería http://localhost:8080/rest/test/hello.

Si todo ha funcionado correctamente, deberíamos ver en el navegador las palabras “Hello World”.

6.2 Entorno de producción

Dando por supuesto que se dispone de un servidor web Tomcat instalado en la máquina en la que se va a desplegar la aplicación web (uno de los requisitos iniciales del proyecto), el proceso de subida a producción es bastante sencillo. Simplemente habrá que utilizar Eclipse para empaquetar el código en forma de aplicación web (formato .war), y transferir dicho fichero al directorio deploy de la instalación de Tomcat correspondiente. Una vez hecho esto, se debe reiniciar el servidor para que la aplicación web se despliegue y pase a estar disponible en el dominio definido en el servidor.

Pasos:

- Empaquetar el código en forma de aplicación web
    - Click derecho en el proyecto; seleccionar la opción Exportar

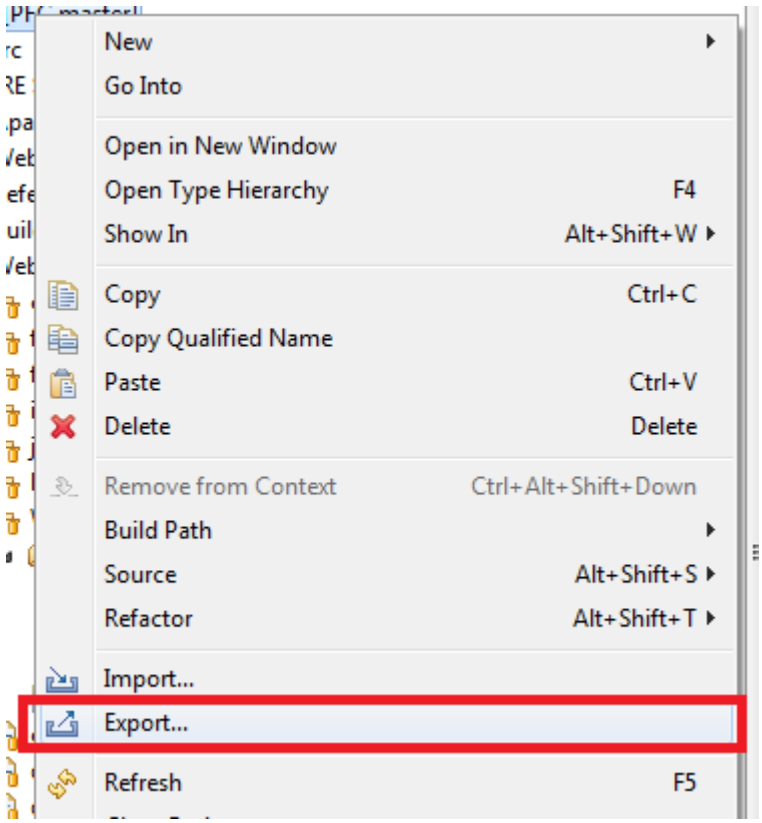


Figura 25: Exportar proyecto en Eclipse

- Exportar como fichero WAR:

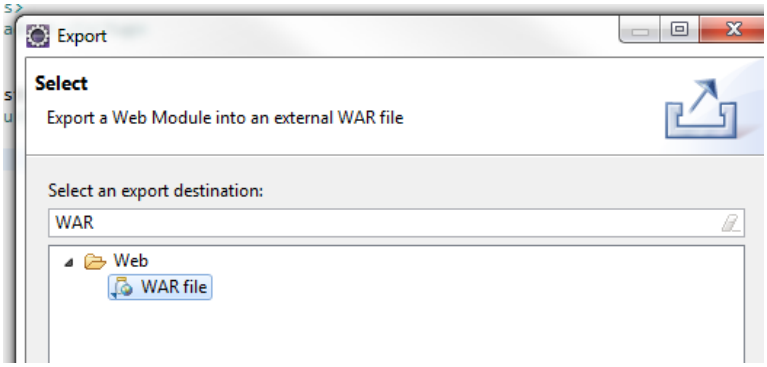


Figura 26: Crear librería en Eclipse

- Transferir el fichero resultante al directorio *deploy* de la instalación de Tomcat correspondiente.

6.3 Validación de la aplicación

Con el objetivo de validar la herramienta y así cumplir con uno de los requisitos establecidos en la definición del proyecto, se procede a cargar los datos del proyecto “Trilogy 2”, en el que el Departamento de Ingeniería Telemática de la Universidad Carlos III actúa como coordinador.

Dicho proyecto consta de cuatro workpackages:

- WP1 - Creating Liquidity
- WP2 - Tussle over Liquidity
- WP3 - Using Liquidity
- WP4 - Dissemination and standardization

Y forman parte de él once partners (se detallan el nombre de la institución y la abreviatura, utilizada como nombre de usuario en la herramienta):

- Universidad Carlos III de Madrid (uc3m)
- British Telecommunications (bt)
- NEC Europe (nec)
- University College London (ucl-uk)
- Universite Catholique de Louvain (ucl-be)
- Intel Software Development (intel)
- Universitatea Politehnica din Bucuresti (upb)
- Nextworks (nxw)
- Onapp (onapp)
- The Chancellor, Masters and Scholars of the University of Cambridge (ucam)
- Telefonica Investigacion y Desarrollo (tid)

Además, se definen períodos de informe trimestrales.

Se han introducido todos estos datos en la herramienta, mostrándose a continuación las capturas de pantalla más importantes de las distintas vistas:

6.3.1 Definición del proyecto: workpackages

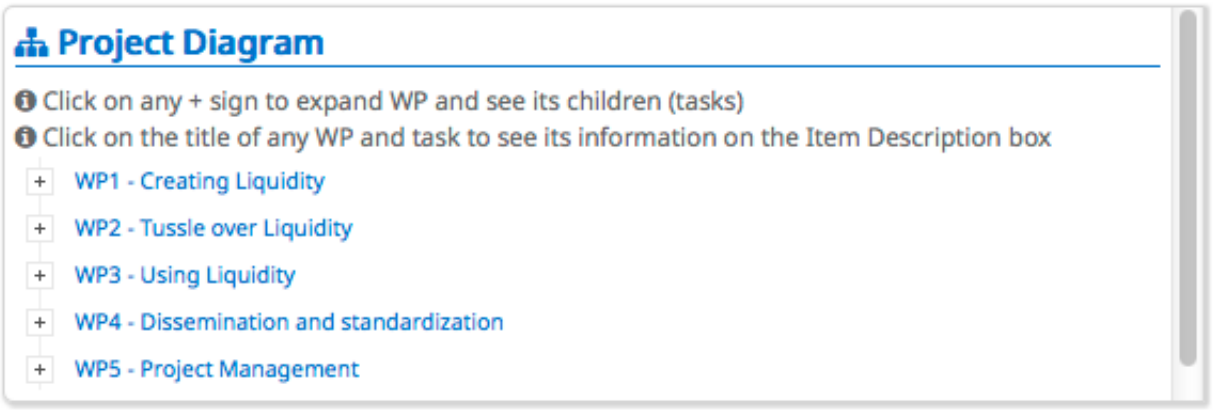


Figura 27: Trilogy 2: Definición del proyecto

6.3.2 Detalle de un Workpackage: WP5 – Project Management

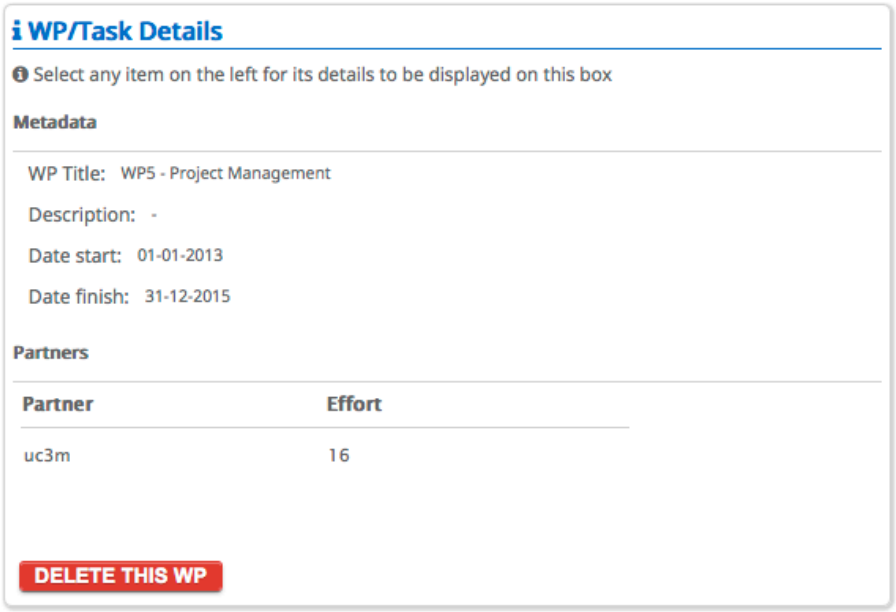
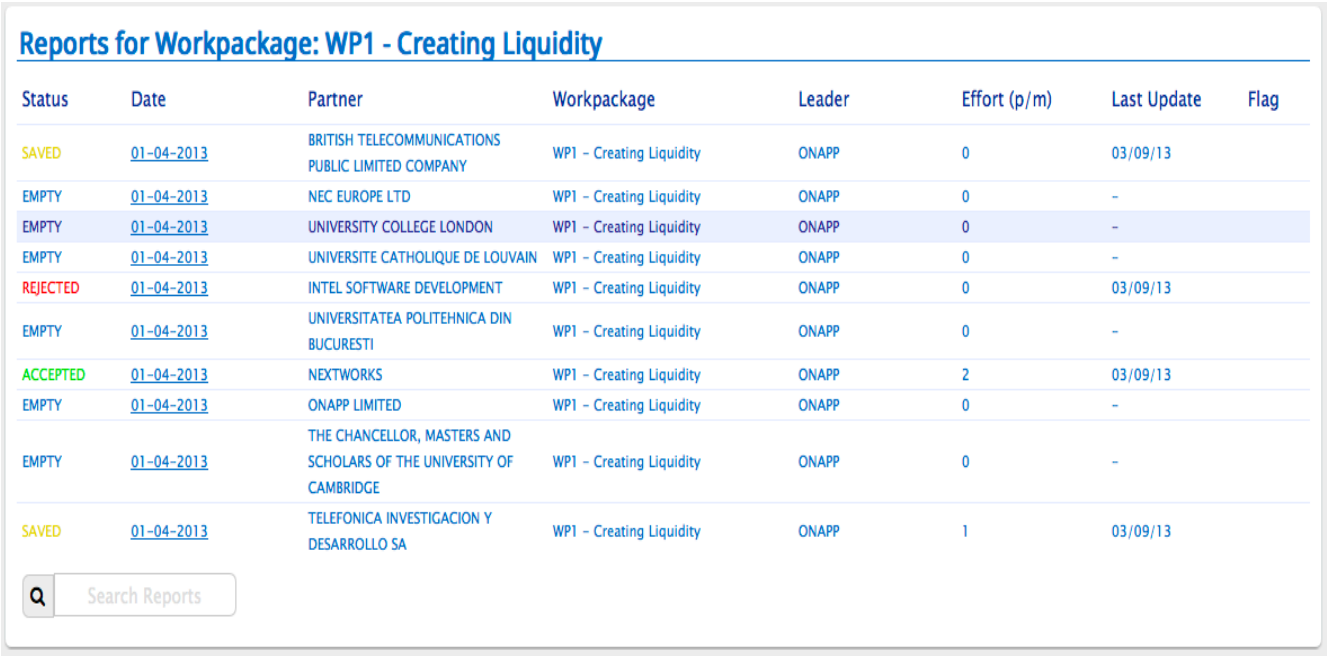


Figura 28: Trilogy 2: Detalles del WP5

6.3.3 Estado del proyecto: WP1



Reports for Workpackage: WP1 - Creating Liquidity

Status	Date	Partner	Workpackage	Leader	Effort (p/m)	Last Update	Flag
SAVED	01-04-2013	BRITISH TELECOMMUNICATIONS PUBLIC LIMITED COMPANY	WP1 - Creating Liquidity	ONAPP	0	03/09/13	
EMPTY	01-04-2013	NEC EUROPE LTD	WP1 - Creating Liquidity	ONAPP	0	-	
EMPTY	01-04-2013	UNIVERSITY COLLEGE LONDON	WP1 - Creating Liquidity	ONAPP	0	-	
EMPTY	01-04-2013	UNIVERSITE CATHOLIQUE DE LOUVAIN	WP1 - Creating Liquidity	ONAPP	0	-	
REJECTED	01-04-2013	INTEL SOFTWARE DEVELOPMENT	WP1 - Creating Liquidity	ONAPP	0	03/09/13	
EMPTY	01-04-2013	UNIVERSITATEA POLITEHNICA DIN BUCURESTI	WP1 - Creating Liquidity	ONAPP	0	-	
ACCEPTED	01-04-2013	NEXTWORKS	WP1 - Creating Liquidity	ONAPP	2	03/09/13	
EMPTY	01-04-2013	ONAPP LIMITED	WP1 - Creating Liquidity	ONAPP	0	-	
EMPTY	01-04-2013	THE CHANCELLOR, MASTERS AND SCHOLARS OF THE UNIVERSITY OF CAMBRIDGE	WP1 - Creating Liquidity	ONAPP	0	-	
SAVED	01-04-2013	TELEFONICA INVESTIGACION Y DESARROLLO SA	WP1 - Creating Liquidity	ONAPP	1	03/09/13	

Search Reports

Figura 29: Trilogy 2: Estados de los informes para el WP1

6.3.4 Detalle de un Informe: Telefonica I+D en WP1

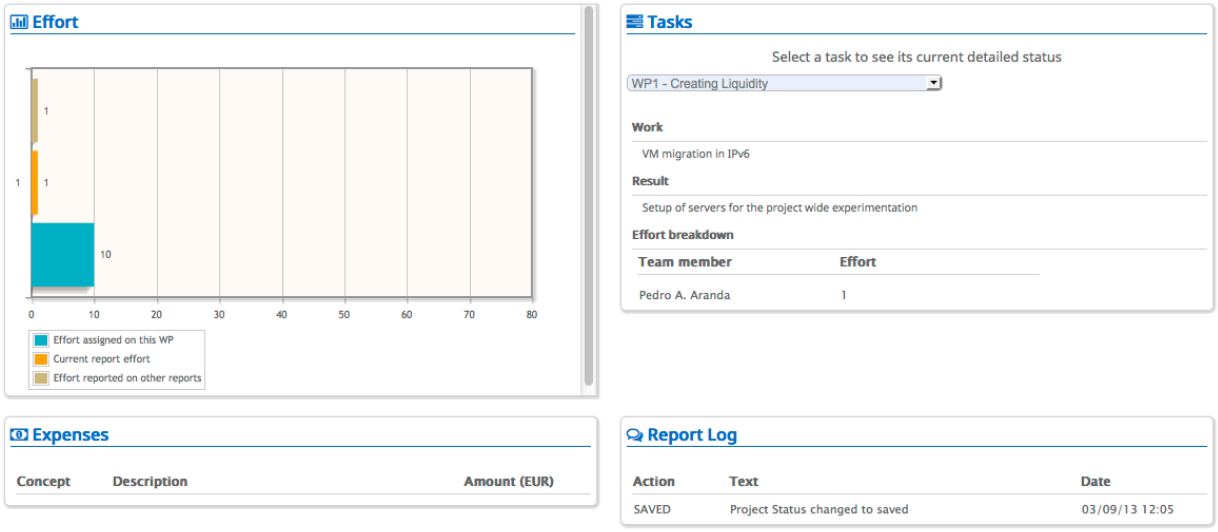


Figura 30: Trilogy 2: Detalles de un informe de trabajo

7 CONCLUSIONES Y TRABAJOS FUTUROS

7.1 Conclusiones

El desarrollo de esta aplicación satisface todos y cada uno de los requisitos presentados en los capítulos iniciales de esta memoria, habiendo aportado además algunas de las más modernas técnicas de Gestión de Proyectos y tecnologías de vanguardia en la fase de implementación y desarrollo.

El hecho de utilizar las recomendaciones del PMBOK especialmente en las fases de definición es un hecho diríamos que diferencial, puesto que en este tipo de Proyectos de Fin de Carrera, usualmente no existe un método de trabajo que marque las pautas del trabajo a realizar. Cierto es que, aún así, hay dos aspectos que para futuros proyectos de este mismo tipo han de ser corregidos:

7.1.1 Plazos

Se hace imprescindible la definición de plazos para las distintas iteraciones del proyecto, de manera que tanto desarrollador como patrocinador del proyecto tengan un calendario al que aferrarse y que les guíe para decidir qué funcionalidades añadir (o desechar) en siguientes iteraciones.

7.1.2 Dedicación

Quizá este sea el aspecto que más margen de mejora ha tenido durante todo el proceso. Como se destacó anteriormente, el hecho de que las dos personas involucradas en el proyecto estén separadas físicamente por más de 6.000 kilómetros y vivan en zonas horarias que difieren 6 horas, hace que la comunicación entre ellos sea clave (siguiente punto), y que además, el desarrollador deba poner de su parte más de lo normal para seguir con una línea aceptable de trabajo. Son muchas las circunstancias que han llevado a que esto así no sea, por tanto la solución que presento para futuros trabajos es precisamente no llegar a este punto, sino que todos los Proyectos de Fin de Carrera sean realizados a tiempo completo, lo que redundaría en una mejora notable de la productividad y eficiencia en los procesos.

7.1.3 Feedback

Cayendo este aspecto a mejorar totalmente en la parte del desarrollador, es cierto que se han llevado a cabo muchos avances y decisiones que no han sido compartidas con el patrocinador en tiempo real, si bien no es menos cierto que ninguna de ellas ha supuesto contratiempo alguno en el desarrollo del proyecto.

7.1.4 Tecnologías

Por increíble que parezca, en el período que va desde la asignación del proyecto (Julio de 2012) hasta su finalización (Agosto de 2013), las tecnologías utilizadas han quedado no vamos a decir obsoletas, pero sí que han visto cómo algunas otras más modernas se han puesto en la vanguardia del desarrollo web. Así por ejemplo, a día de hoy no es aconsejable utilizar desarrollo Javascript en forma de "código espagueti", es decir, funciones sin una estructura de clases u organización de código predefinida. El avance en frameworks Javascript ha sido impresionante en los últimos meses, y si bien la concepción inicial de este proyecto casa perfectamente con el mencionado desarrollo en forma de "código espagueti", ahora se podrían volver a pensar algunas de las ideas iniciales que logren dar al código una forma más moderna (aunque cabe destacar que a nivel funcional y de rendimiento sería complicado mejorar esta implementación actual).

Por tanto, a nivel funcional, podría considerarse este un proyecto completo, si bien todos los aspectos descritos que podrían haber mejorado el proceso de desarrollo y por consiguiente mejorar el producto final en términos productivos en lo relativo al trabajo empleado.

7.2 Trabajos futuros

7.2.1 Mejoras en el código Javascript

Al hilo de uno de los puntos anteriores, cabe destacar el hecho de que el código Javascript puede ser mejorado en términos de estructura y facilidad de comprensión mediante el uso de algún framework actualmente en uso. De entre ellos, destaca por uso y soporte a través de su comunidad Backbone.js. Y es que, a pesar de las facilidades que algunas librerías como jQuery proporcionan a la hora de manipular elementos de una página web de forma dinámica, como dijimos antes, una gran cantidad de código mal estructurado puede convertirse en lo que se conoce como código espagueti. Con jQuery, podemos llegar a tener en código una cantidad inmanejable de selectores, lo que puede suponer que mantener sincronizados la interfaz con el estado de la aplicación y los datos pueda convertirse en una tarea harto complicada.

Backbone.js se aplica para intentar la resolución de algunos estos problemas, definiendo un marco de trabajo (framework) en el que organizar el código. Backbone.js en esencia en una librería modelo-vista-controlador (MVC) para Javascript. Por definición, en este patrón de diseño cada parte es totalmente independiente de las demás, de modo que el modelo mantiene el estado de la aplicación y genera eventos al ser modificado para que así la vista pueda actualizarse de una forma autónoma e independiente para ambos. A su vez, la vista actúa como interfaz del controlador, a través del que se modifica el modelo de forma adecuada, y así todos los elementos están relacionados, pero cada uno se encarga exclusivamente de su parte.

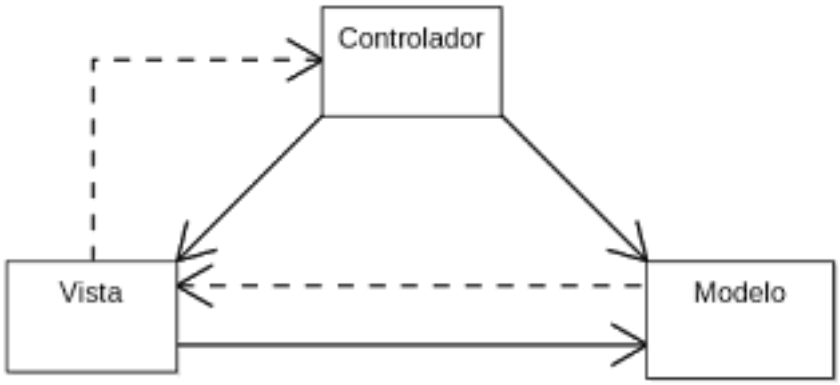


Figura 31: Modelo vista controlador

Backbone además proporciona una forma para trabajar con servicios REST tanto para recuperar los datos de los modelos como para actualizarlos, guardarlos y eliminarlos en el servidor, lo que supone un valor añadido dada la arquitectura de esta aplicación.

7.2.2 Cambio en el almacenamiento de datos

Una de las necesidades iniciales era desarrollar una aplicación portable, en la que no se dependiera de configuración alguna de base de datos a la hora de ser migrada a otro servidor o plataforma. La solución, siguiendo los requerimientos del patrocinador del proyecto, no fue otra que utilizar ficheros XML como método de almacenamiento de datos.

La verdad es que en términos de rendimiento y escalabilidad, los fichero XML se adecúan a la perfección a los requerimientos mínimos de cada una de estos aspectos, si bien es cierto que los actuales servicios en la nube podrían aportar una solución si cabe más robusta.

En este caso, mi propuesta sería migrar el desarrollo de la parte de backend (lógica de negocio, acceso a datos y almacenamiento de datos) a Google App Engine. Este servicio implementa una forma bastante moderna de almacenar la información; mediante bases de datos NO-SQL, en las que cada nueva entrada es creada (y será posteriormente recuperada) a través de un objeto Java. De esta manera, el resultado de la aplicación no es que se haga portable, es que estaría desacoplado a un servidor específico desde un primer momento. Para cubrir la necesidad de disponer de la información de proyectos e informes en un formato físico más manejable, se debería crear una funcionalidad extra que permitiera exportar todos los datos en distintos formatos.

7.2.3 Expandiendo la aplicación: dispositivos móviles

Una de las grandes ventajas de la arquitectura multicapas utilizada en esta aplicación, es el hecho de que se pueden crear múltiples interfaces de cliente que funcionen de manera simultánea. En el mundo de hoy, en el que teléfonos y tabletas van ganando cuota de mercado en lo que acceso a la red se refiere día tras día, esto supone una gran oportunidad para desarrollar aplicaciones adaptadas a estos dispositivos. Los ejemplos más evidentes son aplicaciones para sistemas operativos iOS y/o Android; dado que esta es una aplicación RESTful, cuyos recursos son accesibles mediante simples peticiones HTTP, sería interesante estudiar la posibilidad de dar el salto al mundo móvil, adaptándose a las exigencias y limitaciones de dichas plataformas.

7.3 Tiempo y costes

7.3.1 Desglose del tiempo invertido

Se muestra a continuación el tiempo “bruto” empleado en cada grupo de tareas; no se incluyen aquellas relativas a gestión de proyecto puesto que han sido consideradas dentro de cada una de las demás.

- Definición del proyecto: 3 semanas
- Definición de requisitos funcionales: 2 semanas
- Diseño: 7 semanas.
- Lógica de negocio: 8 semanas.
- Interfaz web: 14 semanas.
- Documentación: 4 semanas.
- Configuración: 4 semanas.

Todo ello resulta en un tiempo total (bruto) de 44 semanas; como se ha mencionado anteriormente en esta memoria, la dedicación no ha sido a tiempo completo, y resulta difícil detallar dicha dedicación por cada tarea; es por ello que se ha determinado un factor aproximado, de 4:1, es decir, la dedicación al proyecto ha sido de un cuarto de lo que se considera una jornada laboral estándar. Por tanto, estas 44 semanas se convierten en 11 en términos de dedicación total, lo que hace un total de 55 días o 440 horas de trabajo efectivo.

7.3.2 Coste del proyecto

7.3.2.1 Costes materiales

1. Ordenador: 1.100 €
2. Viaje Nueva York - Madrid para la presentación del proyecto: 800 €

7.3.2.2 Costes de recursos humanos

Teniendo en cuenta las tarifas actuales para un Ingeniero Técnico con 7 años de experiencia, y haciendo una estimación aproximada, ya que las distintas tareas realizadas tienen unas tarifas que varían de una a otra, hemos dado con una cifra de 50€/hora, con lo que el coste total es de 22.200€.

7.3.2.3 Coste total

Teniendo en cuenta los dos epígrafes anteriores, el coste total del proyecto a efectos de mercado hubiera ascendido a 23.100 €.

8 Bibliografía

Páginas web

[1] Build RESTful Web services and dynamic Web applications with the multi-tier architecture
http://public.dhe.ibm.com/software/dw/ajax/wa-aj-multitier2/wa-aj-multitier2-pdf.pdf

[2] Extensible Markup Language – Wikipedia
http://es.wikipedia.org/wiki/Extensible_Markup_Language

[3] Definiciones de términos
http://www.pergaminovirtual.com.ar/definicion

[4] Javascript Tutorial
http://www.w3schools.com/js/

[5] Representational State Transfer (REST)
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[6] Tutorial: Installing Tomcat 7 and Using it with Eclipse
http://www.coreservlets.com/Apache-Tomcat-Tutorial/tomcat-7-with-eclipse.html

[7] REST: Jersey configuration on Tomcat
http://www.suryasuravarapu.com/2009/02/rest-jersey-configuration-on-tomcat.html

[8] Java MD5 Hashing Example
http://www.mkyong.com/java/java-md5-hashing-example/

[9] jQuery Tutorial
http://www.w3schools.com/jquery/

[10] Learning Center
http://learn.jquery.com/

[11] Easy Java/XML integration with JDOM, Part 2
http://www.javaworld.com/jw-07-2000/jw-0728-jdom2.html

[12] JavaScript md5
http://phpjs.org/functions/md5/

[13] Ajax tutorial
http://www.w3schools.com/ajax/

[14] jqPlot Usage
http://www.jqplot.com/docs/files/usage-txt.html

[15] Font Awesome Examples
http://fortawesome.github.io/Font-Awesome/examples/

[16] Manual de CSS3
http://www.desarrolloweb.com/manuales/css3.html

[17] HTML5 Tutorial
http://www.html-5-tutorial.com/

[18] Time, cost, quality - and knowledge
http://www.nickmilton.com/2013/05/time-cost-quality-and-knowledge.html

[19] Project Management Body of Knowledge
http://es.wikipedia.org/wiki/Project_Management_Body_of_Knowledge

[20] Backbone.js
http://documentcloud.github.io/backbone/

Libros

[21] A Guide to the Project Management Body of Knowledge, Fourth Edition
Project Management Institute

Recursos utilizados en casos de dudas/consultas

[22] StackOverFlow
http://stackoverflow.com/

[23] jQuery forum
https://forum.jquery.com/

ANEXO I – API REST

En este apartado se describe en detalle el API REST creado en esta la aplicación. Como se puede ver, muchas de las URLs definidas requieren de parámetros en la propia URL, que pueden ser complementados por parámetros de formulario, que son el resultado de hacer una llamada POST con dichos parámetros adjuntos (cada plataforma envía estos parámetros de forma distinta; jQuery los codifica en una cadena parámetro=valor en la que los distintos pares son separados por el signo '&').

Todos los parámetros son interpretados como texto, y sólo algunos (que se especifican) tienen que tener ciertos valores determinados de antemano.

Asimismo, se adjunta una pequeña descripción del resultado de cada acción.

/report/get/{projectid}

@PathParam("projectId") String projectId

Devuelve el fichero XML con todos los informes de este proyecto.

/report/addexpenses /{projectId}/{wpId}/{partnerId}/{reportDate}

@FormParam("concept")
@FormParam("description")
@FormParam("amount")
@PathParam("projectId")
@PathParam("wpId")
@PathParam("partnerId")
@PathParam("reportDate")

Añade una nueva entrada de gastos al subinforme

/report/deleteexpenses/{projectId}/{wpId}/{partnerId}/{reportDate}

@FormParam("concept")
@FormParam("description")
@FormParam("amount")
@PathParam("projectId")
@PathParam("wpId")
@PathParam("partnerId")
@PathParam("reportDate")

Elimina la correspondiente entrada entrada de gastos del subinforme

/report/edit/{projectId}/{wpId}/{partnerId}/{reportDate}

@FormParam(“id”)
@FormParam(“value”)
@PathParam(“projectId”)
@PathParam(“wpId”)
@PathParam(“partnerId”)
@PathParam(“reportDate”)

Para el proyecto {projectId}, edita el informe identificado mediante el WP, partner, y fecha de entrega; los campos de formulario id y value determinan qué elemento del informe se va a editar.
Posibles valores de id: comment|explanation|feeddback|status|flag

/task/edit/{projectId}/{wpId}/{partnerId}/{reportDate}/{taskId}

@FormParam(“id”)
@FormParam(“value”)
@PathParam(“projectId”)
@PathParam(“wpId”)
@PathParam(“partnerId”)
@PathParam(“reportDate”)
@PathParam(“taskId”)

Para el proyecto {projectId}, edita la información relativa al trabajo realizado en una determinada tarea en el informe identificado mediante el WP, partner, y fecha de entrega; los campos de formulario id y value determinan qué elemento de la tarea se va a editar.
Posibles valores de id: wrok|result|miembro del grupo (para casos en los que se añade esfuerzo)

/report/sendbyemail/{projectId}/{wpId}/{partnerId}/{reportId}/{email}

@PathParam("email")
@PathParam("projectId")
@PathParam("wpId")
@PathParam("partnerId")
@PathParam("reportDate")

El subinforme en cuestión es enviado por correo electronico a la dirección especificada en el parámetro {email}.

/partners

Devuelve el fichero contenedor de partners

/partners/add

@FormParam("id")
@FormParam("name")

@FormParam("email")
@FormParam("members")
@FormParam("action")
@FormParam("password")

Añade un nuevo partner a la herramienta. El parámetro “action” puede tener los valors “add” o “edit”, según se quiera eliminar la versión anterior del partner (“edit”) o crear una nueva entrada (“add”).

/partners/login

@FormParam("loginid")
@FormParam("token")
@FormParam("n")

Esta es la operación menos intuitiva, puesto que los parámetros token y n no tienen significado aparente; el primero no es otra cosa que la contraseña, codificada en MD5, y el segundo es una marca de tiempo (milisegundos transcurridos desde el 1 de enero de 1970) que deberá ser verificada en el servidor.
Si la combinación login-password es incorrecta y/o la marca de tiempo es aceptada, la respuesta a esta llamada devuelve una excepción y el usuario no puede acceder a la herramienta.

/partners/forgot

@FormParam("loginforgot")
@FormParam("emailforgot")

Si el nombre de usuario y dirección de email proporcionados coinciden con los que están especificados para este usuario en la herramienta, la contraseña se reenvía por correo electrónico a dicha dirección.

/projects

Devuelve el fichero contenedor de proyectos

/project/{projectid}

Devuelve el fichero XML que contiene la información del proyecto {projectid}

/project/create

@FormParam("title")

@FormParam("dateStart")
@FormParam("dateFinish")
@FormParam("description")
@FormParam("coordinator")

Crea un nuevo proyecto con la información recibida en los parámetros de formulario.

/project/add/schedule/{projectId}

@PathParam("projectId")
@FormParam("dateSchedule")

Añade una fecha de entrega de informe al proyecto

/project/add/partner/wp/{projectId}/{wpId}

@PathParam("projectId")
@PathParam("wpId")
@FormParam("partnerWP")
@FormParam("effortWP")

Los partners especificados en el parámetro de formulario son asignados a este WP; se incluye el valor del esfuerzo asignado a este partner

/project/add/partner/task/{projectId}/{wpId}/{taskId}

@PathParam("projectId")
@PathParam("wpId")
@PathParam("taskId")
@FormParam("partnersTask")

Los partners especificados en el parámetro de formulario son asignados a esta tarea

/project/add/WP/{projectId}

@PathParam("projectId")
@FormParam("titleWP")
@FormParam("descriptionWP")
@FormParam("dateInitWP")
@FormParam("dateFinishWP")
@FormParam("coordinatorWP")

Añadde un WP al projecto cuya D se corrrsponde al valor especificado en el parámetro de URL

/project/add/task/{projectId}/{wpId}

@PathParam("projectId")
@PathParam("wpId")
@FormParam("titleTask")
@FormParam("dateInitTask")
@FormParam("dateFinishTask")
@FormParam("partnersTask")
@FormParam("descriptionTask")

Añade una tarea al proyecto con toda la información de los parámetros de formulario

/project/notify/{projectId}

@PathParam("projectId")

Recorre el proyecto en busca de fechas de entrega de informe, notificando a los partners de que han de enviar dichos informes.

/project/{projectId}

@PathParam("projectId")

Devuelve la información del proyecto en formato XML.

/project/delete/task/{projectId}/{taskId}

@PathParam("projectId")
@PathParam("taskId")

Elimina la tarea del proyecto en cuestión.

/project/delete/wp/{projectId}/{wpId}

@PathParam("projectId")
@PathParam("wpId")

Elimina el WP del proyecto en cuestión.